
UNIVERSIDADE FEDERAL DA PARAÍBA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

LEONARDO MEDEIROS

**MÓDULO IP DE UM DEMULTIPLEXADOR PARA O
SUBSISTEMA FLUXO DE TRANSPORTE – MPEG-2 TS –
ADERENTE AO SISTEMA BRASILEIRO DE TV DIGITAL**

JOÃO PESSOA
MARÇO, 2010

LEONARDO MEDEIROS

**MÓDULO IP DE UM DEMULTIPLEXADOR PARA O
SUBSISTEMA FLUXO DE TRANSPORTE – MPEG-2 TS –
ADERENTE AO SISTEMA BRASILEIRO DE TV DIGITAL**

Dissertação submetida ao Programa de Pós-Graduação em Informática da Universidade Federal da Paraíba como exigência parcial para obtenção do grau de Mestre em Informática.

Orientador:
Professor Doutor Antonio Carlos Cavalcanti

Área de concentração: Sistemas de Computação

Linha de Pesquisa: Processamento de Sinais e Sistemas Gráficos

JOÃO PESSOA
MARÇO, 2010

LEONARDO MEDEIROS

**MÓDULO IP DE UM DEMULTIPLEXADOR PARA O
SUBSISTEMA FLUXO DE TRANSPORTE – MPEG-2 TS –
ADERENTE AO SISTEMA BRASILEIRO DE TV DIGITAL**

Banca Examinadora:

Professor Doutor **Rômulo Pires Coelho Ferreira**

Professora Doutora **Tatiana Aires Tavares**

Professor Doutor **Lucidio dos Anjos Formiga Cabral**

Professor Doutor **José Antonio Gomes de Lima**

Professor Doutor **Antonio Carlos Cavalcanti** (orientador)

JOÃO PESSOA

MARÇO, 2010

M488m Medeiros, Leonardo.
 Módulo IP de um Demultiplexador para o Subsistema Fluxo
de Transporte- MPEG-2-Aderente ao Sistema Brasileiro de TV
Digital/ Leonardo Medeiros. – João Pessoa, 2010.
 139f .
 Orientador: Antonio Carlos Cavalcanti.
 Dissertação (Mestrado) – UFPb - CCEN

 1.Informática. 2. Demultiplexador. 3. Circuito Integrado –TV
Digital.

UFPb/BC

CDU: 004 (043)

Ata da Sessão Pública de Defesa de Dissertação de
Mestrado de **Leonardo Medeiros**, candidato ao
Título de Mestre em Informática na Área de
Sistemas de Computação, realizada em 04 de
março de 2010.

Aos quatro dias do mês de março do ano dois mil e dez, às 14 horas e 30 minutos, no Auditório do Centro de Ciências Exatas e da Natureza da Universidade Federal da Paraíba, reuniram-se os membros da Banca Examinadora constituída para examinar o candidato ao grau de Mestre em Informática, na área de "Sistemas de Computação", na linha de pesquisa "Processamento de Sinais e Sistemas Gráficos", o Sr. Leonardo Medeiros. A comissão examinadora foi composta pelos professores doutores: Antonio Carlos Cavalcanti (DI-UFPB), Orientador e Presidente da Banca Examinadora, Lucidio dos Anjos Formiga Cabral (DI-UFPB), Jose Antonio Gomes de Lima (DI-UFPB), Tatiana Aires Tavares (DI-UFPB) examinadores internos e Rômulo Pires Coelho Ferreira (Instituto Federal de Educação, Ciência e Tecnologia de Alagoas-IFAL), examinador externo. Dando início aos trabalhos, o professor Antonio Carlos Cavalcanti, cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato para que o mesmo fizesse, oralmente, a exposição do trabalho de dissertação intitulado "Módulo IP de um Demultiplexador para o Subsistema Fluxo de Transporte-MPEG2-TS Aderente ao Sistema Brasileiro de Televisão Digital". Concluída a exposição, o candidato foi argüido pela Banca Examinadora que emitiu o seguinte parecer: "aprovado". Assim sendo, deve a Universidade Federal da Paraíba expedir o respectivo diploma de Mestre em Informática na forma da lei e, para constar, eu, professora Tatiana Aires Tavares, Coordenadora deste Programa, servindo de secretária lavrei a presente ata que vai assinada por mim mesmo e pelos membros da Banca Examinadora. João Pessoa, 04 de março de 2010.

Tatiana Aires Tavares
Tatiana Aires Tavares

Prof. Dr. Antonio Carlos Cavalcanti
Orientador (DI-UFPB)

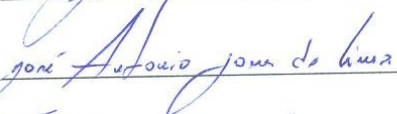
Prof. Dr. Lucidio dos Anjos Formiga Cabral
Examinador Interno (DI-UFPB)

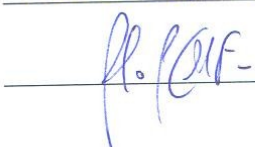
Prof. Dr. José Antonio Gomes de Lima
Examinador Interno (DI-UFPB)

Profa. Dra. Tatiana Aires Tavares
Examinador Interno (DI-UFPB)

Prof. Dr. Rômulo Pires Coelho Ferreira
Examinador Externo (IFAL)







Agradecimentos

Agradeço primeiramente a Deus, que apesar da minha pouca fé sempre esteve ao meu lado me dando força e coragem nas horas mais difíceis, dando forças para concluir esse trabalho, indicando sempre o caminho correto a seguir, entrego então todos os meus sonhos e projetos em suas mãos.

À minha mãe que sempre esteve ao meu lado, doando todo seu amor e força, sendo um exemplo de mãe e de pessoa, que mesmo com os contratempos da vida, nunca perdeu a amabilidade e dedicação aos filhos.

Ao meu pai que me apoiou em tudo na minha vida, sempre desejando minha felicidade; onde diante do seu caráter e simplicidade, tornou-se o meu grande exemplo de vida e é, sobretudo, o meu melhor amigo em todos os sentidos.

Aos meus irmãos e familiares que sempre torceram por mim e pelo meu sucesso, me dando todo amor e carinho possível.

Aos meus grandes amigos, que nunca deixaram de dar um sorriso ou um abraço quando precisei e que hoje são verdadeiros irmãos; espero sempre compartilhar a vida com essas pessoas tão especiais que partilhei momentos inesquecíveis, repletos de companheirismo, amor e alegria.

Aos meus mestres, em especial ao meu orientador Antonio, que além de ser uma grande professor, de sua orientação e de seus ensinamentos, mostrou-se ser um verdadeiro amigo, sendo o apoio essencial para a pesquisa. Agradeço imensamente seu apoio e incentivo durante toda minha estadia no LASIC.

Ao CNPq pelo apoio financeiro.

Resumo

No cenário brasileiro, o desenvolvimento de componentes de hardware através de metodologias e ferramentas computacionais, disponibilizados a custos acessíveis a centros de ensino e pesquisa tornou factível a possibilidade de se projetar até o nível de propriedade intelectual – IP, blocos e módulos de circuitos integrados para atender áreas estratégicas da indústria, em especial aquelas voltadas aos componentes de produtos aderentes ao Sistema Brasileiro de Televisão Digital (SBTVD). Este trabalho trata o desenvolvimento e validação de uma proposta arquitetural para o Subsistema de Fluxo de Transporte MPEG-2 TS – demultiplexador de fluxo de transporte – do SBTVD, desde sua especificação e modelagem em alto-nível, até sua implementação no nível de mapeamento tecnológico. Também estão apresentados resultados específicos como o detalhamento dos requisitos estruturais e funcionais do Subsistema de Fluxo de Transporte – MPEG-2 TS, o desenvolvimento de um modelo em software para o demultiplexador e de um ambiente contendo fluxos MPEG-2 TS capazes de validar a conformidade dos modelos com a especificação do SBTVD, a implementação e validação de uma prova de conceito para um demultiplexador MPEG-2 TS no nível RTL e sua prototipagem com mapeamento tecnológico para FPGA e para um design kit de ASIC.

Palavras-chave: Demultiplexador. Circuito Integrado. IP. MPEG-2 TS. SBTVD.

Abstract

In the Brazilian scenario, the development of hardware components through computational methodologies and tools, available at affordable costs to education and research centers have make it feasible to design blocks and integrated circuit modules up to intellectual property – IP level, focusing strategic areas of industry, specially those dedicated to the components for the Brazilian Digital Television System (SBTVD). This work deals with the architectural proposal development and validation for the Transport Stream MPEG-2 TS Subsystem of SBTVD – transport stream demultiplexer – from its specification and high-level modeling to its technology mapping implementation. As specific results, the structural and functional requirements of Transport Stream Subsystem – MPEG-2 TS are detailed, the development of a software model for the demultiplexer and of an environment containing MPEG-2 TS flows that can validate the compliance with the SBTVD specification of models, an RTL-level MPEG-2 TS demultiplexer IP conceptual proof implementation and validation and its prototyping with technology mapping for FPGA and an ASIC design kit , are also presented.

Key-words: Demultiplexer. Integrated Circuit. IP. MPEG-2 TS. SBTVD.

Lista de Figuras

Figura 1: Níveis de abstração e domínios de projeto (WEST, HARRIS, 2005).....	25
Figura 2: Fluxo de projeto em forma de Y apresentada por Gajki e Kuhn (WEST, HARRIS, 2005).....	27
Figura 3: Arquitetura de hardware típica de um SoC (WAGNER, CESARIO, CARRO, JERRAYA, 2004).....	34
Figura 4: Modelo de processo de projeto de um sistema (KEATING, BRICAUD, 2002).	36
Figura 5: Fluxo de projeto e de verificação (WEST, HARRIS, 2005).....	44
Figura 6: Arquitetura do Sistema Brasileiro de Televisão Digital.....	47
Figura 7: Requisitos básicos da arquitetura do receptor (ABNT NBR 15604, 2007).....	48
Figura 8: Diagrama de blocos geral da arquitetura de referência para o receptor.....	49
Figura 9: Fluxo de dados através do demultiplexador.....	51
Figura 10: Abordagem básica de multiplexação para fluxos elementares de áudio e vídeo do padrão MPEG-2 Sistema (ISO/IEC 13818-1, 2000).....	52
Figura 11: Composição do fluxo de transporte – TS (INAMORI, NAGANUMA, WAKABAYASHI, ENDO, 1996).....	53
Figura 12: Composição do fluxo de programa – PS (INAMORI, NAGANUMA, WAKABAYASHI, ENDO, 1996).....	53
Figura 13: Pacote de Fluxo de Transporte.....	54
Figura 14: Fluxo de Transporte e Seções PSI/SI (HANINTAK, 2008).....	55
Figura 15: Diagrama de blocos do SAA7205H (1997).....	59
Figura 16: Diagrama de blocos do IBM39MPEGCS24PFA16C e IBM39MPEGCS24DPFA16C (1999).....	60
Figura 17: Fluxo de dados do TS através do demultiplexador presente nos chipsets IBM39MPEGCS24PFA16C e IBM39MPEGCS24DPFA16C (1999).....	62
Figura 18: Diagrama de blocos do SAA7214 (2001).....	63
Figura 19: Diagrama de blocos do SAA7219 (2001).....	64
Figura 20: Diagrama de blocos do SAA7240 (2001).....	65
Figura 21: Diagrama de blocos do S5H2010 (2003).....	66
Figura 22: Diagrama de blocos do demultiplexador presente no chipset S5H2010 (2003).....	67
Figura 23: Diagrama de blocos do STB7100 (2005).....	68
Figura 24: Subsistema de fluxo de transporte do STB7100 (2005).....	69
Figura 25: Diagrama de blocos do STI7101 (2008).....	70
Figura 26: Tipos de pacote TS.....	73
Figura 27: Demultiplexador de referência.....	76
Figura 28: Diagrama do bloco demultiplexador destacando os sinais de entrada e saída.....	80
Figura 29: Modelo hierárquico da especificação executável do demux.....	81
Figura 30: Aplicação do fluxo de transporte da validação na especificação de alto nível do demultiplexador proposto.....	83
Figura 31: Fluxo de transporte usado na validação.....	84
Figura 32: Fluxo de transporte apresentado através da ferramenta MPEG TS Utils.....	85
Figura 33: Grafo de sequenciamento ordenado das operações do demultiplexador proposto.....	89
Figura 34: Grafo de fluxo de dados do Demultiplexador.....	90
Figura 35: Microarquitetura do demultiplexador proposto.....	92
Figura 36: Estados e transições da máquina de estados finita do TS_PACKET_FILTER.	96

Figura 37: Estados e transições da máquina de estados finita do PROGRAM_PACKET_FILTER.....	97
Figura 38: Estados e transições da máquina de estados finita do TRANSPORT_PACKET_ANALYSER.....	98
Figura 39: Estados e transições da máquina de estados finita do TRANSPORT_PACKET_HEADER_ANALYSER.....	98
Figura 40: Estados e transições da máquina de estados finita do ADAPTATION_FIELD_ANALYSER.....	99
Figura 41: Estados e transições da máquina de estados finita do PCR_FILTER.....	100
Figura 42: Estados e transições da máquina de estados finita do PES_HEADER_ANALYSER.....	101
Figura 43: Estados e transições da máquina de estados finita do PTS_DTS_FILTER.....	102
Figura 44: Estados e transições da máquina de estados finita do PSI_SI.....	104
Figura 45: Estados e transições da máquina de estados finita do PSI_SI_HEADER_ANALYSER.....	105
Figura 46: Estados e transições da máquina de estados finita do PAT_FILTER.....	106
Figura 47: Estados e transições da máquina de estados finita do PMT_FILTER.....	107
Figura 48: Estados e transições da máquina de estados finita do SDT_FILTER.....	107
Figura 49: Estados e transições da máquina de estados finita do CRC_32_DETECTOR.....	108
Figura 50: Exemplo de cobertura funcional realizada nos blocos.....	116
Figura 51: Comparação dos arquivos de saída de vídeo da especificação de alto nível e da implementação RTL.....	118
Figura 52: Comparação dos arquivos de saída de áudio da especificação de alto nível e da implementação RTL.....	118
Figura 53: Relatório de área do ts_demux.....	122
Figura 54: Simulação pós-síntese da netlist do TS_DEMUX para Stratix II.....	123

Lista de Tabelas

Tabela 1: Três classificações de IP cores baseado em cinco critérios (PALMA, MORAES, CALAZANS, 2001).....	38
Tabela 2: Quadro comparativo dos chips comerciais apresentados nesta dissertação.....	71
Tabela 3: Interfaces do demultiplexador.....	75
Tabela 4: Componentes do plano de verificação funcional do demultiplexador proposto.....	110
Tabela 5: Características verificadas do bloco ts_packet_filter.....	111
Tabela 6: Características verificadas do bloco program_packet_filter.....	111
Tabela 7: Características verificadas do bloco in_fifo.....	112
Tabela 8: Características verificadas do bloco in_mem.....	112
Tabela 9: Características verificadas do bloco transport_packet_header_analyser.....	113
Tabela 10: Características verificadas do bloco adaptation_field_analyser.....	113
Tabela 11: Características verificadas do bloco pcr_filter.....	113
Tabela 12: Características verificadas do bloco pes_header_analyser.....	114
Tabela 13: Características verificadas do bloco pts_dts_filter.....	114
Tabela 14: Características verificadas do bloco psi_si_header_analyser.....	114
Tabela 15: Características verificadas do bloco pat_filter.....	114
Tabela 16: Características verificadas do bloco pmt_filter.....	115
Tabela 17: Características verificadas do bloco sdt_filter.....	115
Tabela 18: Características verificadas do bloco crc_32_detector.....	115
Tabela 19: Critérios de conclusão da verificação funcional do demultiplexador.....	116
Tabela 20: Recursos utilizados na verificação funcional do demultiplexador.....	117
Tabela 21: Informações de síntese em FPGA do demultiplexador.....	120
Tabela 22: Informações de área para as tecnologias ASIC disponíveis no design kit ADK 3.1.....	121
Tabela 23: Informações de utilização do dispositivo EP1S10F780C.....	125
Tabela 24: Informações de utilização do dispositivo EP1S60F1020C.....	126
Tabela 25: Informações de utilização do dispositivo EP2S60F672C.....	126
Tabela 26: Informações de utilização do dispositivo EP3C25F324C.....	127
Tabela 27: Relatório de tempos de setup da netlist do ts_demux para Stratix II.....	136
Tabela 28: Relatório de tempos de hold da netlist do ts_demux para Stratix II.....	137
Tabela 29: Relatório de tempos de relógio para saída da netlist do ts_demux para Stratix II.....	138
Tabela 30: Relatório de tempos mínimos de relógio para saída da netlist do ts_demux para Stratix II.....	139

Lista de Abreviaturas

ABNT – Associação Brasileira de Normas Técnicas

ADK – ASIC Design Kit

AHB – Advanced High-performance Bus

AMBA – Advanced Microcontroller Bus Architecture

APB – AMBA Peripheral Bus

ARIB – Association of Radio Industries and Businesses

ARM – Advanced RISC Machine

ASIC – Circuito Integrado de Aplicação Específica (Application-Specific Integrated Circuit)

AXI – Advanced eXtensible Interface

BAT – Tabela de Associação de Buquê (Bouquet Association Table)

BFM – Modelo Funcional de Barramento (Bus Functional Model)

BIT – Tabela de Informação do Radiodifusor (Broadcaster Information Table)

CAD – Projeto guiada por computador (Computer-Aided Design)

CAS – Sistema de Acesso Condicional (Conditional Access System)

CAT – Tabela de Acesso Condicional (Conditional Access Table)

CEITEC - Centro de Excelência em Tecnologia Eletrônica Avançada

CI – Circuito Integrado

CPU – Unidade Central de Processamento (Central Processing Unit)

CRC – Verificação de redundância cíclica (Cyclic Redundancy Check)

CRC-32 – Verificação de redundância cíclica usando polinômios de 32 bits (Cyclic Redundancy Check using 32 bits polynomial)

D-VCR – Digital Video Cassette Recorder

DDR – Dupla Taxa de Transferência (Double-Data-Rate)

DEMO - Demonstração

DMA – Acesso Direto à Memória (Direct Memory Access)

DSP – Processador Digital de Sinais (Digital Signal Processor)

DTS – Marca de Tempo de Decodificação (Decoding Time Stamp)

DUV – Design Under Verification

DVB – Transmissão de Vídeo Digital (Digital Video Broadcasting)

E/S – Entrada/Saída

ECM – Mensagens de Controle de Autorização (Entitlement Control Message)

EIT – Tabela de Informação de Eventos (Event Information Table)

EMM – Mensagens de Gerenciamento de Autorização (Entitlement Management

Message)

EPG – Guia Eletrônico de Programação (Electronic Program Guide)

ERT – Tabela de Relação de Eventos (Event Relation Table)

ES – Fluxo Elementar (Elementary Stream)

ESCR - Referência de Relógio de Fluxo Elementar (Elementary Stream Clock Reference)

ETSI – European Telecommunications Standards Institute

FIFO – Estrutura de dados do tipo Fila (First In First Out)

FINEP - Financiadora de Estudos e Projetos

FPGA – Matriz de portas de campo programável (Field-Programmable Gate Array)

FSM – Máquina de Estados Finita (Finite-State Machine)

GB – Giga Byte

gcc – GNU Compiler Collection

GP/HS – Propósito Geral/Alta Velocidade (General Purpose/High Speed)

HD – Alta Definição (High Definition)

HDL – Linguagem de Descrição de Hardware (Hardware Description Language)

HEP - High Education Program

IBM – International Business Machines

ID – Identificador (Identifier)

IEC – International Electrotechnical Commission

IEEE - Instituto de Engenheiros Eletricistas e Eletrônicos (Institute of Electrical and Electronics Engineers)

IP – Propriedade Intelectual (Intellectual Property)

ISO – International Organization for Standardization

ITT – Tabela de Transmissão de Índice (Index Transmission Information Table)

LASIC – Laboratório de Arquitetura, Sistemas Integráveis e Circuitos

LASID – Laboratório de Sistemas Digitais

LDT – Tabela de Referência de Outras Tabelas (Linked Description Table)

LIT – Tabela de Informação de Evento Local (Local Event Information Table)

LUT - Look up Table

MCT – Ministério de Ciência e Tecnologia

MIPS – Microprocessador sem Estágios Interligados de Pipeline (Microprocessor without Interlocked Pipeline Stages)

MP@HL - MainProfile@HighLevel

MSP – Processador de Sistema MPEG-2 (MPEG-2 System Processor)

NBIT – Tabela de Informação de Grupo da Rede (Network Board Information Table)

NBR – Norma Brasileira

NIT – Tabela de Informação de Rede (Network Information Table)

nMOS – Metal óxido-semicondutor do tipo n (n-type metal-oxide-semiconductor)

NoC – Rede em Chip (Network-on-Chip)

NOP – Nenhuma Operação (No Operations)

NRSS – Serviços de Suporte Não residencial (Non-Residential Support Services)

NTSC – Comitê Nacional do Sistema de Televisão (National Television System Committee)

OCP – Open Core Protocol

OCP-IP – Open Core Protocol – International Partnership

PAL – Linha de Fase Alternante (Phase Alternating Line)

PAT – Tabela de Associação de Programa (Program Association Table)

PCAT – Tabela de Anúncio de Conteúdo Parcial (Partial Content Announcement Table)

PCM – Modulação por Código de Pulsos (Pulse-Code Modulation)

PCR – Referência de Relógio de Programa (Program Clock Reference)

PES – Packetized Elementary Stream

PID – Identificador de Pacote (Packet Identifier)

PLA – Matriz de lógica programável (Programmable Logic Array)

pMOS – Metal óxido-semicondutor do tipo p (p-type metal-oxide-semiconductor)

PMT – Tabela de Mapa de Programa (Program Map Table)

PS – Fluxo de Programa (Program Stream)

PSI – Informação Específica de Programa (Program Specific Information)

PTI – Interface Programável de Transporte (Programmable Transport Interface)

PTS – Marca de Tempo de Apresentação (Presentation Time Stamp)

PWM – Modulação por largura de pulso (Pulse Width Modulation)

RAM – Memória de Acesso Aleatório (Random Access Memory)

RF – Rádio Frequência

RISC – Computador com um Conjunto Reduzido de Instruções (Reduced Instruction Set Computer)

RST – Tabela de Estado do Evento (Running Status Table)

RTL – Nível de Transferência de Registrador (Register Transfer Level)

S/PDIF – Formato de Interface Digital Sony/Philips (Sony/Philips Digital Interface Format)

SBMicro – Sociedade Brasileira de Microeletrônica

SBTVD – Sistema Brasileiro de Televisão Digital

SD – Definição Padrão (Standard Definition)

SDRAM – Memória de Acesso Aleatório Dinâmica Síncrona (Synchronous Dynamic Random Access Memory)

SDT – Tabela de Descrição de Serviços (Service Description Table)

SI – Serviço de Informação (Service Information)

SLR – Registrador de Deslocamento para a Esquerda (Shift Left Register)

SoC – Sistema em Chip (System-on-Chip)

SoPC – Sistema em Chip Programável (System-on-Programmable-Chip)

SRAM – Memória Estática de Acesso Aleatório (Static Random Access Memory)

ST – Tabela de Preenchimento (Stuffing Table)

STC – Relógio de Tempo do Sistema (System Time Clock)

TAR – Terminal de Acesso de Referência

TDT – Tabela de Data e Horário (Time and Date Table)

TOT – Tabela de Referência de Data e Horário (Time Offset Table)

TS – Fluxo de Transporte (Transport Stream).

TV – Televisão

UFPB – Universidade Federal da Paraíba

ULA – Unidade Lógica Aritmética

VHDL – Linguagem de Descrição de Hardware de circuitos integrados de velocidade muito alta (Very high speed integrated circuits Hardware Description Language)

VLC - VideoLAN Client

VLSI – Integração em escala muito alta (Very-Large-Scale Integration)

Sumário

RESUMO.....	6
ABSTRACT.....	7
LISTA DE FIGURAS.....	8
LISTA DE TABELAS.....	10
LISTA DE ABREVIATURAS.....	11
SUMÁRIO.....	15
INTRODUÇÃO.....	18
1 DELIMITAÇÃO DO OBJETO E METODOLOGIA.....	21
1.1 Objetivos Gerais.....	21
1.2 Objetivos Específicos.....	21
1.3 Metodologia.....	22
2 FUNDAMENTAÇÃO TEÓRICA.....	24
2.1 Fundamentos de Projeto de Circuitos Integrados Digitais VLSI.....	24
2.1.1 Projeto de Circuitos Integrados.....	25
2.1.2 Métodos de Projeto.....	28
2.1.3 Projeto Baseado em Plataforma – SoC (System-on-Chip).....	31
2.1.4 Projeto no Nível de Sistema.....	37
2.1.5 Projeto no Nível de Arquitetura.....	39
2.1.6 Projeto no Nível de Microarquitetura.....	41
2.1.7 Projeto no Nível RTL.....	42
2.1.8 Estratégias de Verificação.....	43
2.1.9 Medidas de Qualidade.....	45
2.2 Sistema Brasileiro de TV Digital – SBTVD.....	46
2.2.1 Arquitetura Básica de um Terminal de Acesso.....	48
2.2.2 Decodificação do fluxo de entrada.....	50
2.2.3 Demultiplexador de MPEG-2 Sistema – MPEG-2 TS.....	50
2.2.4 O Padrão MPEG-2 Sistema.....	51
2.2.5 Visão Geral do Pacote TS.....	53
2.2.6 Infraestrutura PES.....	55
2.2.7 Infraestrutura PSI/SI.....	56
2.3 Evolução de Chips Comerciais MPEG-2 TS.....	58
2.3.1 SAA7205H (1997) – Philips.....	58
2.3.2 IBM39MPEGCS24PFA16C e IBM39MPEGCS24DPFA16C (1999) – IBM.....	60

2.3.3 SAA7214 (2001) – Philips	63
2.3.4 SAA7219 (2001) – Philips.....	64
2.3.5 SAA7240 (2001) – Philips.....	65
2.3.6 S5H2010 (2003) – Samsung.....	66
2.3.7 STB7100 (2005) – ST Microelectronics.....	67
2.3.8 STI7101 (2008) – ST Microelectronics.....	70
2.3.9 Resumo dos Chips Comerciais apresentados.....	71
3 ESPECIFICAÇÃO, MODELAGEM E VALIDAÇÃO EM ALTO NÍVEL DO DEMULPLEXADOR PROPOSTO.....	72
3.1 Especificação do Demultiplexador Proposto.....	72
3.1.1 Especificação funcional.....	73
3.1.2 Especificação arquitetural.....	76
3.1.3 Especificação de verificação.....	78
3.2 Implementação do modelo funcional hierárquico de alto-nível.....	78
3.3 Implementação dos testbenches de verificação para o modelo de alto-nível.....	82
4 ARQUITETURA DO DEMULPLEXADOR PROPOSTO.....	86
4.1 Visão Comportamental no Nível Arquitetural do Demultiplexador.....	86
4.1.1 Recursos.....	86
4.1.2 Restrições.....	88
4.1.3 Grafo de sequenciamento.....	88
4.1.4 Grafo de Fluxo de Dados.....	90
4.2 Microarquitetura do Demultiplexador.....	92
5 IMPLEMENTAÇÃO RTL E MAPEAMENTO TECNOLÓGICO DO DEMULPLEXADOR PROPOSTO.....	94
5.1 Visão Comportamental no Nível RTL.....	94
5.1.1 IN_BUFFER.....	95
5.1.2 TRANSPORT_PACKET_ANALYSER.....	97
5.1.3 PES_EXTRACTOR.....	100
5.1.4 PSI_SI_EXTRACTOR.....	104
5.2 Verificação Funcional do Demultiplexador.....	109
5.2.1 Características Verificadas por Bloco.....	110
5.2.2 Requisitos de Cobertura e de Conclusão da Verificação.....	115
5.2.3 Recursos Utilizados.....	116
5.3 Validação no nível RTL.....	117
5.4 Sítese lógica e mapeamento tecnológico.....	119

5.4.1 FPGA.....	119
5.4.2 ASIC.....	120
5.4.3 Validação no nível das portas (gate level).....	122
6 RESULTADOS DE MAPEAMENTO E AVALIAÇÃO.....	125
6.1 Resultados de Mapeamento Tecnológico em FPGA.....	125
6.2 Avaliação dos Resultados Obtidos.....	127
CONSIDERAÇÕES FINAIS.....	128
REFERÊNCIAS BIBLIOGRÁFICAS.....	130
ANEXO A.....	136

Introdução

Os esforços recentes da política governamental e da inteligência brasileira lograram êxito na definição de um padrão brasileiro de TV digital, o Sistema Brasileiro de Televisão Digital – SBTVD. Ao contemplar os últimos avanços mundiais em todas tecnologias e possibilidades de serviços e negócios envolvidos e ao introduzir aspectos originais inovadores, tais esforços abrem grandes oportunidades, em particular para ramo de Projeto de Circuitos Integrados.

O terminal de acesso, como parte do sistema de recepção do sistema de televisão digital, possui o objetivo de receber o sinal digital de televisão e exibi-lo ao usuário. A configuração do receptor é baseada em um sintonizador, um processador, módulos dedicados à decodificação dos sinais digitais e/ou software para realizar essa função e prover a interface com o usuário. Internacionalmente, os receptores mais recentes, se baseiam nos chamados *System-on-Chip* – SoC, que barateiam o custo e aumentam o desempenho do terminal de acesso ao integrarem funções que exigiriam processamento sofisticado em software (consequentemente processadores complexos) em componentes de *hardware*.

Dentre os processos importantes do sistema de recepção que vêm sendo implementados em *hardware* está a decodificação do sinal digital em suas múltiplas camadas. Destas, tem especial destaque a camada de entrada no processo de decodificação: a camada de transporte do nível sistema do sinal digital codificado. Neste nível o sinal codificado está na forma de fluxo de transporte (TS – *transport stream*) definido pelo padrão internacional MPEG-2 Sistema (ISO/IEC 13818-1, 2000) e usado pelo SBTVD. O TS é resultado da multiplexação dos sinais de áudio, vídeo, dados e informações específicas de programas.

Para sua camada de transporte, o SBTVD definiu os padrões ABNT NBR 15602-3 (2007), ABNT NBR 15603-1 (2007), ABNT NBR 15603-2 (2007) e ABNT NBR 15603-3 (2007).

Dá-se o nome de demultiplexação à tarefa de decodificar a camada sistema do TS, ou seja a desmontagem do sinal multiplexado, enviando cada fluxo elementar (áudio, vídeo, etc) ao seu respectivo destino: decodificadores de áudio e vídeo digital, aplicações

embarcadas, etc. Demultiplexador é então a denominação do módulo responsável por esta função.

Um sistema eletrônico integrado para o demultiplexador pode ser projetado e implementado na forma de uma propriedade intelectual, ou simplesmente IP (*Intellectual Property*). O desenvolvimento de um IP para este módulo visa fornecer opção de componente a ser integrado em SoCs para terminais de acesso do SBTVD.

É possível agregar valor nas diversas etapas de projeto e implementação de sistemas eletrônicos integrados. Embora seja óbvia a dimensão do volume de capital envolvido com a etapa de produção dos componentes e integração física dos sistemas, é nas etapas anteriores, sobretudo na exploração de patentes e **licenciamentos de uso de propriedade intelectual**, onde vai residir a maior parcela da agregação de valor.

Com a entrada em funcionamento realizada no dia 5 de fevereiro de 2010 da unidade de fabricação do CEITEC (Centro de Excelência em Tecnologia Eletrônica Avançada), o produto final, ou seja, o chip em silício, dependendo da escala e tecnologia, também pode ser produzido no Brasil. De todo modo, quando os volumes são do porte do mercado mundial, ou mesmo do brasileiro de telefonia sem fio e TV digital, apenas plantas industriais em poucos países do mundo estão em posição de responder à demanda.

Por outro lado, criou-se no Brasil um ecossistema propício para o desenvolvimento de projetos de circuitos integrados. No nível acadêmico, com o Programa Nacional de Microeletrônica (bolsas de mestrado e doutorado) e através de metodologias e ferramentas computacionais de nível industrial internacional disponibilizadas através do MCT/FINEP/SBMicro “Programa Software para as Universidades” a centros de ensino e pesquisa; no nível industrial, com o programa CI Brasil que, além de subsidiar o acesso ao *software*, alavanca a formação massiva de recursos humanos para projeto de circuitos integrados e sua colocação diretamente nas *design houses* emergentes apoiadas pelo programa.

Diante de tal cenário, buscou-se explorar duas vertentes estratégicas:

- desenvolver uma prova de conceito de um circuito integrado passível de se transformar em um módulo de propriedade intelectual – IP, preferencialmente para um componente utilizável no projeto de receptores compatíveis com o SBTVD;
 - para escolher tal componente, estudar detalhadamente o conteúdo digital presente
-

na especificação brasileira para o fluxo de transporte (TS – *Transport Stream*), com vistas a identificar as oportunidades de processamento desses conteúdos em hardware.

Como resultado desse trabalho preliminar, o componente escolhido foi o demultiplexador do fluxo de transporte: processador do MPEG-2 TS.

Para descrever o trabalho realizado, esta dissertação está organizada da seguinte forma:

No capítulo 1 o objeto de estudo é delimitado e é apresentada a metodologia utilizada.

O capítulo 2 apresenta os fundamentos teóricos relacionados ao projeto de circuitos integrados, com foco no método de projeto baseado em plataforma – SoC (WEST, HARRIS, 2005), desde o seu projeto em alto-nível até o nível de transferência de registrador (RTL – *Register Transfer Level*).

Aqui, o sistema de recepção do SBTVD com a arquitetura de referência brasileira do receptor são apresentados; o processo de decodificação do sinal de TV digital é discutido, a infraestrutura necessária de um demultiplexador é detalhada e os blocos funcionais do subsistema de recepção são descritos. Ao fim desse capítulo, são apresentadas organizações internas de SoCs comerciais que implementam essas funções.

O capítulo 3 desse trabalho é dedicado à especificação, modelagem e validação do demultiplexador, com a construção de um modelo funcional em software e *testbenches* de verificação em níveis hierárquicos.

No capítulo 4, é apresentado o nível de arquitetura do demultiplexador proposto, seu modelo comportamental e microarquitetura.

O capítulo 5 apresenta a implementação RTL do demultiplexador proposto juntamente com sua verificação funcional, seguindo-se seu mapeamento tecnológico para prototipagem em FPGA.

Por fim, o capítulo 6 expõe e avalia os resultados obtidos.

Encerra-se a dissertação com as considerações finais a respeito do trabalho realizado e trabalhos futuros são propostos.

1 Delimitação do Objeto e Metodologia

Este trabalho trata do desenvolvimento e validação de uma proposta arquitetural para o Subsistema de Fluxo de Transporte MPEG-2 TS – demultiplexador de fluxo de transporte – do Sistema Brasileiro de TV Digital, desde sua especificação e modelagem alto-nível, até sua implementação no nível RTL.

1.1 Objetivos Gerais

Os objetivos gerais são:

- Aprofundar os conhecimentos relativos às estruturas de multiplexagem/demultiplexagem, codificação/decodificação dos fluxos de dados e sinais que serão tratados no padrão SBTVD;
- Capacitar-se, no nível de mestrado, para o estado da arte no fluxo de projeto de um circuito integrado complexo, baseado nos blocos capazes de processar o fluxo de transporte do SBTVD.

1.2 Objetivos Específicos

Os objetivos específicos são:

- Detalhar os requisitos estruturais e funcionais do Subsistema de Fluxo de Transporte – MPEG-2 TS – em conformidade com o SBTVD;
 - Desenvolver modelo em software para o demultiplexador MPEG-2 TS e um conjunto de fluxos TS capaz de validar sua conformidade com a especificação do SBTVD;
 - Implementar e validar prova de conceito de um IP do demultiplexador no nível RTL;
 - Prototipagem do demultiplexador com mapeamento tecnológico em FPGA;
 - Prototipagem do demultiplexador com mapeamento tecnológico em ASIC.
-

1.3 Metodologia

Para atingir os objetivos propostos a metodologia utilizada consistiu nas tarefas a seguir:

1. Estudar e detalhar os requisitos de interfaceamento, software e hardware do Subsistema de Fluxo de Transporte-MPEG-2 TS da arquitetura de referência do terminal de acesso do Sistema Brasileiro de TV digital-SBTVD;
 2. Estudar componentes arquiteturais de implementações comerciais recentes de demultiplexadores MPEG-2 TS;
 3. Construir um fluxo de transporte (TS) com conteúdos que servirão à validação dos diversos modelos e níveis que serão produzidos;
 4. Desenvolver e validar em software um modelo funcional hierárquico (especificação executável) de um demultiplexador com seus blocos e sub blocos, que atenda aos requisitos levantados. Para tanto foi utilizada a seguinte sequência de atividades preconizadas por De Micheli (1994):
 - Definição das operações, dependências e sequenciamento;
 - Definição dos recursos funcionais disponibilizados;
 - Definição do conjunto de restrições;
 - Construção do grafo de sequenciamento;
 5. Montar ambiente de verificação das implementações dos próximos níveis de modelagem em linguagem SystemVerilog;
 6. Criar, a partir do modelo em software devidamente validado, um modelo RTL em linguagem Verilog sintetizável e validá-lo no ambiente de verificação com o fluxo de transporte que validou o modelo em software;
 7. Mapear com síntese lógica implementação(ões) de prova de conceito para plataforma de hardware (re)configurável e/ou design kit(s) adequado(s);
 8. Produzir fluxo de teste com parâmetros de desempenho temporal;
 9. Simular, validar e avaliar desempenho da(s) implementação(ões) no mesmo ambiente.
-

A infra estrutura utilizada é detalhada a seguir:

- Computador com processador Pentium 4, 1 GB de memória, 80 GB de disco e Sistema Operacional CentOS 5.x;
 - Notebook Intel Dual Core, 1 GB de memória, Windows Vista;
 - Compilador gcc 3.4.4 (GCC, 2008);
 - Simulador: Mentor Graphics QuestaSim 6.4 (QUESTA, 2009);
 - Linguagens: C com biblioteca Mentor Graphics *Algorithmic C Datatypes* – v1.2 – (DATATYPES, 2008), Verilog e SystemVerilog;
 - Síntese lógica: Mentor Graphics Precision RTL Plus – v2008a.39 – (PRECISION RTL PLUS, 2009) e LeonardoSpectrum – Level 3 v2008a.5 (LEONARDOSPECTRUM, 2009);
 - Síntese lógica, análise de temporização e *backannotation*: Altera Quartus II 8.0 (ALTERA, 2010).
-

2 Fundamentação Teórica

Neste capítulo são abordadas questões teóricas relativas ao projeto VLSI (*Very Large Scale Integration*) e ao SBTVD. Assim, é proporcionado um melhor entendimento das questões introduzidas nos próximos capítulos com relação ao problema tratado por esta dissertação. São apresentadas formas de se projetar um circuito integrado e os seus vários níveis de detalhamento. Apresenta, também, uma visão geral sobre o processo de recepção e de decodificação MPEG-2 Sistema do SBTVD. Por fim, é apresentada uma evolução de chips comerciais que envolvem o fluxo de transporte.

2.1 Fundamentos de Projeto de Circuitos Integrados Digitais VLSI

Projetistas de circuitos integrados – CIs – têm desenvolvido e adotado estratégias para a formação de um conjunto coerente de princípios para ampliar, em tempo, a probabilidade de sucesso de seus projetos. Estilos de projeto e ferramentas de projeto guiadas por computador (CAD – *Computer-Aided Design*) estão envolvidas nos avanços da tecnologia e aumento de produtividade (WEST, HARRIS, 2005).

Existem quatro estágios para a criação de um circuito integrado (DE MICHELI, 1994): **projeto, fabricação, teste e empacotamento**. O estágio de projeto é dividido em três tarefas principais: **conceitualização e modelagem, síntese e otimização, e validação**. A primeira tarefa consiste em distribuir uma ideia em um modelo, que captura a função que o circuito vai realizar. Síntese consiste em refinar o modelo, de um mais abstrato para um mais detalhado, que possui todas as características necessárias para a fabricação. Validação consiste em verificar a consistência dos modelos usados durante o projeto, assim como algumas propriedades do modelo original.

Modelos de circuitos são usados para representar as ideias dos projetistas. A **modelagem** desempenha a parte principal de um projeto microeletrônico, porque representa o veículo usado para transportar informação. A modelagem precisa ser rigorosa, assim como geral, transparente para o projetista e legível por máquina. Para a modelagem podem ser usadas linguagens de descrição de hardware – HDL (*Hardware Description Language*) – assim como modelos gráficos como diagramas de fluxo,

componentes. Especifica a conexão de componentes necessária para alcançar o comportamento desejado;

- **Domínio Físico** – relaciona-se aos **objetos físicos** de um projeto. Especifica como organizar os componentes a fim de conectá-los.

Por sua vez, os níveis de abstração de projeto podem ser:

- **Nível de Especificação Formal** – formado por **modelos formais** de computação, os modelos desse nível são máquinas capazes de fazer tarefas computacionais;
- **Nível de Especificação Funcional** – formado normalmente por **funções**, os modelos desse nível fornecem poucos detalhes, possuindo um alto nível de abstração. Esses modelos não possuem considerações temporais;
- **Nível Arquitetural** – formado normalmente por blocos de hardware, modelos HDL ou diagramas de fluxo, os modelos desse nível são vistos como **componentes interligados** que funcionam em conjunto para realizar uma determinada tarefa. Os modelos desse nível realizam um conjunto de operações, como computação ou transferência de dados;
- **Nível RTL** – formado normalmente por diagramas de **transição de estados**, ULAs e Registradores, os modelos desse nível possuem o propósito de descrição do projeto em um nível menos detalhado que o nível de portas, mas detalhado o suficiente para ser entendido por uma máquina de síntese;
- **Nível Lógico** – formado normalmente por portas e *flip-flops*, os modelos desse nível fornecem o grau mais alto de detalhamento para simulação de circuitos digitais e o nível mais baixo de abstração no domínio digital. Nesse nível, um circuito digital avalia um conjunto de **funções lógicas**;
- **Nível de Circuito** – formado por transistores, os modelos desse nível são vistos, de maneira simplificada, como **chaves**. Os transistores são dispositivos analógicos e podem ser projetados e analisados nesse domínio.
- **Nível Geométrico** – formado normalmente por retângulos, *floor plans* ou leiautes, os circuitos são um conjunto de **entidades geométricas**.

A classificação dos modelos relaciona-se a um conjunto de tarefas de síntese. A

síntese pode ser visto como um conjunto de transformações entre dois domínios (DE MICHELI, 1994).

Por exemplo, a nível arquitetural é possível representar um circuito em seu domínio comportamental como um grafo de sequenciamento. No momento que o grafo de sequenciamento é transformado em uma unidade de controle com um caminho de dados, realiza-se uma **síntese a nível arquitetural**, porque agora se possui um circuito representado no domínio estrutural nesse mesmo nível. Do mesmo modo, quando a nível lógico se possui um circuito representado em seu domínio comportamental como uma máquina de estados finita, e é transformado em um conjunto de portas e *flip-flops* interligadas, se tem a chamada **síntese lógica**.

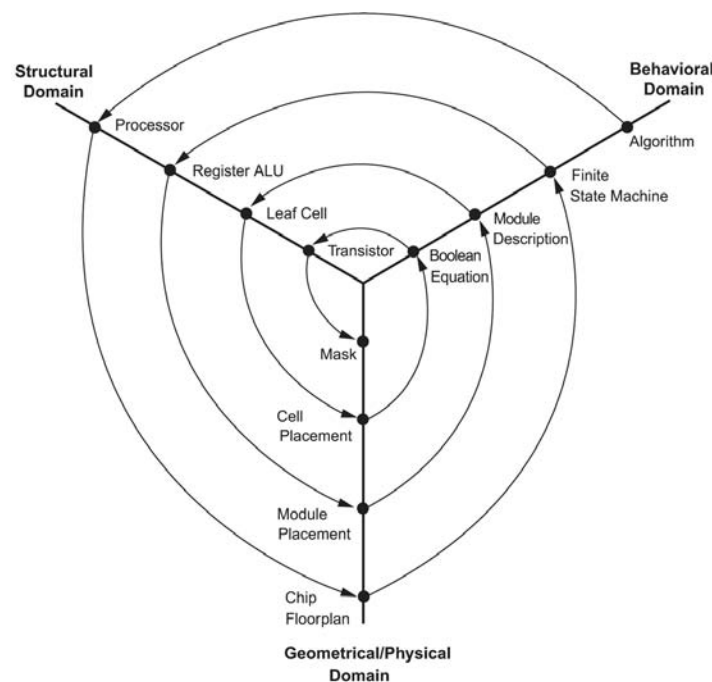


Figura 2: Fluxo de projeto em forma de Y apresentada por Gajki e Kuhn (WEST, HARRIS, 2005).

Um **fluxo de projeto**, ou fluxo de concepção, pode ser entendido como um conjunto de operações que são realizadas em sequencia para a concepção de uma pastilha de circuito integrado. A figura 2 apresenta o fluxo de projeto em forma de Y proposto por Gajki e Kuhn (GAJKI83 *apud* WEST, HARRIS, 2005). O fluxo de projeto permite ao projetista progredir de uma especificação até a implementação final da pastilha. Normalmente, as etapas de síntese são automáticas, embora tenha um

guiamento humano. As etapas de verificação também devem estar presentes no fluxo de projeto (WEST, HARRIS, 2005).

2.1.2 *Métodos de Projeto*

Existe uma variedade de **métodos de projeto** que podem ser usados para implementar um sistema. A descrição dos métodos feita a seguir é a proposta por West e Harris (2005), uma das principais referências de fato na área de projeto de circuitos integrados:

- Microprocessador / DSP

O método mais prático de resolver um problema de projeto de um sistema é usar um microprocessador padrão ou um processador digital de sinais (DSP – *Digital Signal Processor*). Existe uma variedade de microprocessadores disponíveis no mercado, com diferentes velocidades de relógio, tamanhos de memória, etc. Deve-se considerar uma eventual integração quando se decide construir um sistema com um microprocessador.

- Lógica Programável

Pastilhas de circuito integrado programáveis podem ser mais eficientes que microprocessadores de propósito geral e ter o desenvolvimento mais rápido que projetos totalmente personalizados. Este método permite ao projetista avaliar os requisitos de um sistema particular para um CI e recomendar um solução.

Existem pastilhas de circuito integrado com matrizes de lógica programável, com interconexão programável e com interconexão e lógica programável. Os dispositivos de lógica programável – PLA (*Programmable Logic Array*) – consiste em um plano AND (lógica “e”) e um plano OR (lógica “ou”) para calcular qualquer função expressa como soma de produtos. Do tipo matriz de portas de campo programável – FPGA – existem duas versões básicas: a primeira usa a opção de um fusível ou anti-fusível para programar as conexões permanentemente e personalizar a lógica, essa versão é programável apenas uma vez; a segunda usa células estáticas de RAM para personalizar

o roteamento e funções lógicas, consistindo em uma matriz de células lógicas circundada por recursos de roteamento programável.

- Projeto de Arranjo de Circuitos ou Oceano de Circuitos

O projeto de arranjo de circuitos é um método que consiste em construir uma base comum formando uma matriz de transistores – linhas de transistores *nMOS* e *pMOS* – e então personalizar a pastilha alterando a metalização – máscaras de metal e via – que é posicionada acima dos transistores. Este arranjo de transistores não é contínuo. Existe ainda a possibilidade de ter seu tamanho individualizado para implementações econômicas de aplicações específicas, como memórias.

O projeto de oceano de circuitos é uma subclasse do arranjo de circuitos, onde a diferença é que o arranjo de transistores, nesta subclasse, é contínuo.

- Projeto Baseado em Células

O projeto baseado em células usa uma biblioteca de células padrões como base para construir blocos da pastilha de circuito integrado. As células são posicionadas em posições adequadas, e então suas conexões são roteadas. Pode-se ter como resultado, pastilhas menores, mais rápidos e de potência mais baixa que arranjos de circuitos ou lógica programável, mas o custo de fabricação para produzir um conjunto de máscaras personalizadas é maior. É apenas econômico para um grande volume ou quando o desempenho justifica um lucrativo preço de venda.

Comparado ao projeto totalmente personalizado, o projeto baseado em células oferece muito mais produtividade por usar leiautes de células pré-projetadas. *Foundries* e fornecedores de bibliotecas fornecem células com uma grande variedade de funcionalidades. Pode-se escolher da biblioteca as células que entreguem mais corrente, dissipem menor potência. Existem ainda, bibliotecas sofisticadas que geram memórias de tamanhos variados. O rendimento destas memórias pode ser verificado em termos da variação dos tempos de acesso, ciclos de relógio e dissipação de potência.

- Projeto Totalmente Personalizado

Nos fluxos de projeto modernos, muitas visões diferentes são necessárias para integração do projeto. Em um conjunto de visões, por exemplo, a visão temporal pode ser necessária para uma verificação de temporização, a visão lógica para uma simulação e a visão de circuito para comparações *leiaute* versus *esquemática*.

Uma abordagem mais direta é escrever rotinas de posicionamento, que é a essência do posicionamento manual de células padrões. Pode ser preferível em um certo projeto de somador, ter um *leiaute* de caminho de dados específico. Um algoritmo pode ser escrito para posicionar as células na grade de células padrões e um algoritmo de ligação para gerar a *netlist* de portas em HDL. Desse modo, o projeto físico e estrutural são capturados.

- Projeto Baseado em Plataforma – SoC

Como os sistemas digitais integrados têm se tornado mais complexos, o (re)uso de blocos IP tem se tornado frequente. No projeto de sistemas computadorizados, é comum o uso de padrões abertos para blocos como processadores RISC (ARM, MIPS, SPARC,...), geradores de memória, além de funções de entrada e saída conectadas a barramentos aderentes a padrões conhecidos (AMBA, AVALON, ...).

O projeto baseado em plataforma é um método que consiste em desenvolver a solução completa de um sistema pelo uso de um conjunto de blocos aderentes a um padrão (em geral aberto) interconectados, atuando principalmente no problema de particionamento entre hardware (blocos conectados) e software (“rodando” no(s) núcleo(s) RISC). Para implementar um projeto, usam-se IPs de blocos funcionais e **estruturas abertas** de interconexão, no nível do hardware, e linguagens de alto-nível para programar os processadores. Um conjunto coerente de tais componentes constitui uma plataforma.

System-on-Chip baseado em plataforma refere-se ao método de integração de todos os blocos que compõem o sistema em desenvolvimento em um único circuito integrado, usando apenas componentes padrões aderentes a essa plataforma.

Este método se tornou uma tendência, pois traz o benefício da modularidade, que permite a (re)utilização de componentes previamente desenvolvidos e testados pelo

mercado.

Por essas razões, o método de SoC baseado em plataforma norteou o desenvolvimento aqui apresentado e será detalhado na sequência.

2.1.3 Projeto Baseado em Plataforma – SoC (System-on-Chip)

A indústria eletrônica está na era dos chips com bilhões de transistores. A quantidade de transistores de um circuito integrado versus a quantidade de transistores que os engenheiros podem projetar (*capacidade versus produtividade*) estão afetando o tempo de projeto (*time-to-market*), que se torna muito longo, e o custo de projeto (*viabilidade*), se torna muito alto (KEATING, BRICAUD, 2002).

O aumento da complexidade dos sistemas é evidente. Com isso, a capacidade de projetar sistemas complexos em tempo razoável diminui, enquanto a lacuna entre a capacidade e produtividade parece aumentar (GAJSKI, 1999). Dessa forma, o esforço de projeto aumenta e se torna preciso construir em cima da habilidade de outros, para alcançar a demanda de *time-to-market* (THOMAS, 1999).

A exigência está modificando a forma como circuitos VLSI são projetados. *Time-to-market* curto, complexidade alta e alto desempenho é a realidade do mercado de projetos VLSI (PALMA, MORAES, CALAZANS, 2001). O desafio da indústria é continuar entregando dispositivos de forma melhor, mais rápida e mais barata. A solução proposta para esse cenário é aumentar a produtividade (KEATING, BRICAUD, 2002).

A utilização de uma plataforma com um ou mais processadores núcleos e muitos pequenos processadores periféricos é uma alternativa para aumentar a produtividade (GAJSKI, WU, CHAIYAKUL, et al, 2000).

Na abordagem baseada em **plataforma**, as derivações são obtidas pela adaptação de um modelo arquitetural (ou plataforma), que normalmente é de um domínio específico, para os requisitos de um aplicação específica. Esse modelo é baseado em uma plataforma de hardware, consistindo em uma dada estrutura de comunicação e um conjunto de componentes (processadores, memórias, blocos de hardware). O SoC alvo é projetado como uma derivação desse modelo, onde a estrutura de comunicação, os componentes e uma plataforma de software é adaptada para se ajustar aos requisitos da aplicação específica (WAGNER, CESARIO, CARRO, JERRAYA, 2004).

O **reuso de componentes** e partes de sistema também faz sentido, se tornando vital para o crescimento da indústria de semicondutores (GAJSKI, WU, CHAIYAKUL, et al, 2000).

Para ser reusável, um projeto deve ser primeiro usável, isto é, um projeto robusto e correto, projetado para resolver um problema geral, para uso em múltiplas tecnologias, para simulação com uma variedade de simuladores, com interfaces baseadas em padrões, verificado independente do chip em que será usado, com um alto nível de confiança, e totalmente documentado em termos de aplicações apropriadas e restrições (KEATING, BRICAUD, 2002).

Os projetistas devem encontrar e avaliar componentes reusáveis que se ajustam a necessidades específicas. Para que a integração ocorra com sucesso, é necessário que as funcionalidades e interfaces sejam bem definidas. A padronização também é essencial para a promoção do reuso em todas as dimensões do processo de projeto (WAGNER, CESARIO, CARRO, JERRAYA, 2004).

A visão estabelecida para reuso é focada no nível estrutural. Um componente se torna reusável através de interfaces padrões. A interface permite esconder detalhes de projeto de um componente como IP e torna a prática de reuso de componentes possível (SCHAUMONT, CMAR, VERNALDE, et al, 1999).

O projeto baseado em plataforma, dentro de uma perspectiva de reusabilidade, corresponde ao reuso da *expertise* de projetar sistemas. Expressado por um modelo arquitetural, a plataforma facilita o reuso de componentes de hardware e software que se ajustam a esse modelo (WAGNER, CESARIO, CARRO, JERRAYA, 2004).

O reuso de habilidade em diferentes domínios é necessário para o projeto de sistemas heterogêneos, que podem ser completados com êxito através de ambientes de projeto distribuídos e cooperativos (WAGNER, CESARIO, CARRO, JERRAYA, 2004).

Um terceiro modo de aumentar a produtividade é **aumentar nível de abstração** (GAJSKI, 1999) de forma que seja possível uma síntese a partir de uma descrição funcional de alto-nível (GAJSKI, WU, CHAIYAKUL, et al, 2000). Em um cenário onde a produtividade e a qualidade são simultaneamente necessárias, técnicas que iniciem em um alto nível de abstração se tornam cruciais para o processo de projeto (OLIVEIRA, BRISOLARA, WAGNER, CARRO, 2005).

O que se espera com tudo isso, é melhorar a eficiência de projeto e desempenho

de *time-to-market* (THOMAS, 1999).

Componentes de hardware em um SoC incluem um ou vários processadores, mesmo que sejam de diferentes tipos (microcontroladores, DSP, RISC), memórias, componentes dedicados para acelerar tarefas críticas e interfaces para vários periféricos. Os componentes são conectados por redes de comunicação arbitrárias, que podem ser de um simples barramento ou barramentos hierárquicos conectados por pontes até um complexo *network-on-chip* (NoC) (WAGNER, CESARIO, CARRO, JERRAYA, 2004).

As principais diretrizes para projetos SoC são a clara separação entre computação e comunicação, e entre função e arquitetura. Essa distinção fortalece um projeto modular e promove a independência de projeto e a evolução de três aspectos de projeto de um sistema: função, arquitetura e comunicação (WAGNER, CESARIO, CARRO, JERRAYA, 2004).

Diante desta necessidade de comunicação do módulo IP, onde módulos IP se comunicam uns com os outros, a utilização de um **protocolo de comunicação padrão** é recomendado, dada a necessidade de integração. A própria produção de IPs por empresas só tem sentido se elas seguirem determinados padrões, pois se não for assim, se tornaria difícil a comunicação entre eles. Existem esforços direcionados para a padronização de interfaces de comunicação entre os IPs.

Os projetos SoC podem utilizar várias formas de comunicação, entre eles os barramentos. Os componentes se interligam aos barramentos, assim, as empresas podem criar componentes utilizando a interface do barramento. Utilizar barramento, torna desnecessário a criação de protocolos de comunicação personalizados.

Os padrões de comunicação podem ser baseados em algum barramento ou ser independente de barramento. Como exemplo dos baseados em barramentos, podemos citar os padrões AMBA (ARM), CoreConnect (IBM), Avalon (Altera), WISHBONE (Silicore). Como exemplo dos independentes de barramento podemos citar o padrão OCP (OCP-IP). A seguir, é dada especial atenção ao barramento AMBA, devido à sua utilização neste trabalho.

O AMBA é um barramento on-chip de padrão aberto, desenvolvido pela ARM, que tem como principal funcionalidade realizar a interconexão de módulos de um SoC. O AXI é a nova geração do AMBA. Desenvolvido também para sistemas que necessitam alto desempenho e que possuem alta frequência de clock. Possui características como: alta

largura de banda e baixa latência, possibilidade de operações de alta frequência se a utilização de pontes complexas, flexibilidade, compatibilidade com AHB e APB e suporte a transferências fora de ordem.

Cada componente AXI utiliza um único sinal de relógio. Todos os sinais de entrada são amostrados na borda de subida. Todas as alterações de sinais de saída devem ocorrer depois da borda de subida. Não devem existir caminhos combinacionais entre os sinais de entrada e saída tanto nas interfaces mestre como escravo.

O protocolo AXI inclui um simples sinal de reset ativado em *low*. O sinal de reset pode ser declarado de forma assíncrona, mas sua desativação deve ser síncrona depois de uma borda de subida. O protocolo AXI possui uma interface de controle de clock onde é possível entrar e sair de um estado de baixo consumo.

O projeto de um SoC tem início com a definição e validação de uma especificação de alto-nível puramente funcional, que não é influenciada por escolhas arquiteturais e que não consideram como os requisitos de projeto (força, desempenho) serão cumpridos (WAGNER, CESARIO, CARRO, JERRAYA, 2004). A arquitetura típica de um SoC é apresentada na figura 3.

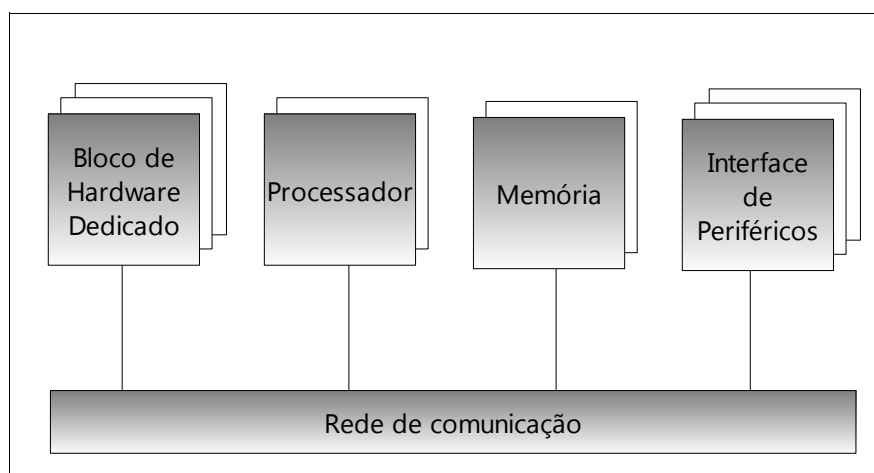


Figura 3: Arquitetura de hardware típica de um SoC (WAGNER, CESARIO, CARRO, JERRAYA, 2004).

A primeira parte de um **processo de projeto** consiste em desenvolver, verificar e refinar um conjunto de especificações até elas estarem detalhadas o suficiente para permitir o início de uma modelagem comportamental seguida de uma modelagem

arquitetural.

As especificações podem ser formais ou informais (MADISETTI, SHEN, 1997) e descrevem o comportamento de um sistema, ou seja, como manipular as interfaces de um sistema para produzir o comportamento desejado (KEATING, BRICAUD, 2002). Tanto propriedades funcionais, quanto temporais e de interface precisam ser descritas.

As especificações funcionais descrevem as interfaces de um sistema ou bloco como visto pelos seus usuários. Descrevem pinos, barramentos, registradores e como podem ser usados para manipular dados. As especificações arquiteturais descrevem as interfaces entre as partes do componente e como as transações dessas interfaces coordenam as funções de cada bloco, e cria o comportamento a nível de sistema desejado (KEATING, BRICAUD, 2002).

Existem duas técnicas principais de especificação: **especificação formal** e **especificação executável**. Na especificação formal, as características desejadas são definidas independentemente de qualquer implementação. As especificações executáveis são, atualmente, mais usadas para descrever o comportamento funcional em muitas situações de projeto. Uma especificação executável é, tipicamente, um modelo abstrato para o hardware que está sendo especificado. Uma especificação de alto-nível é normalmente escrita em C/C++, uma linguagem de verificação de hardware, ou outra linguagem de alto-nível. Uma especificação de baixo nível é escrita em VHDL ou Verilog (KEATING, BRICAUD, 2002).

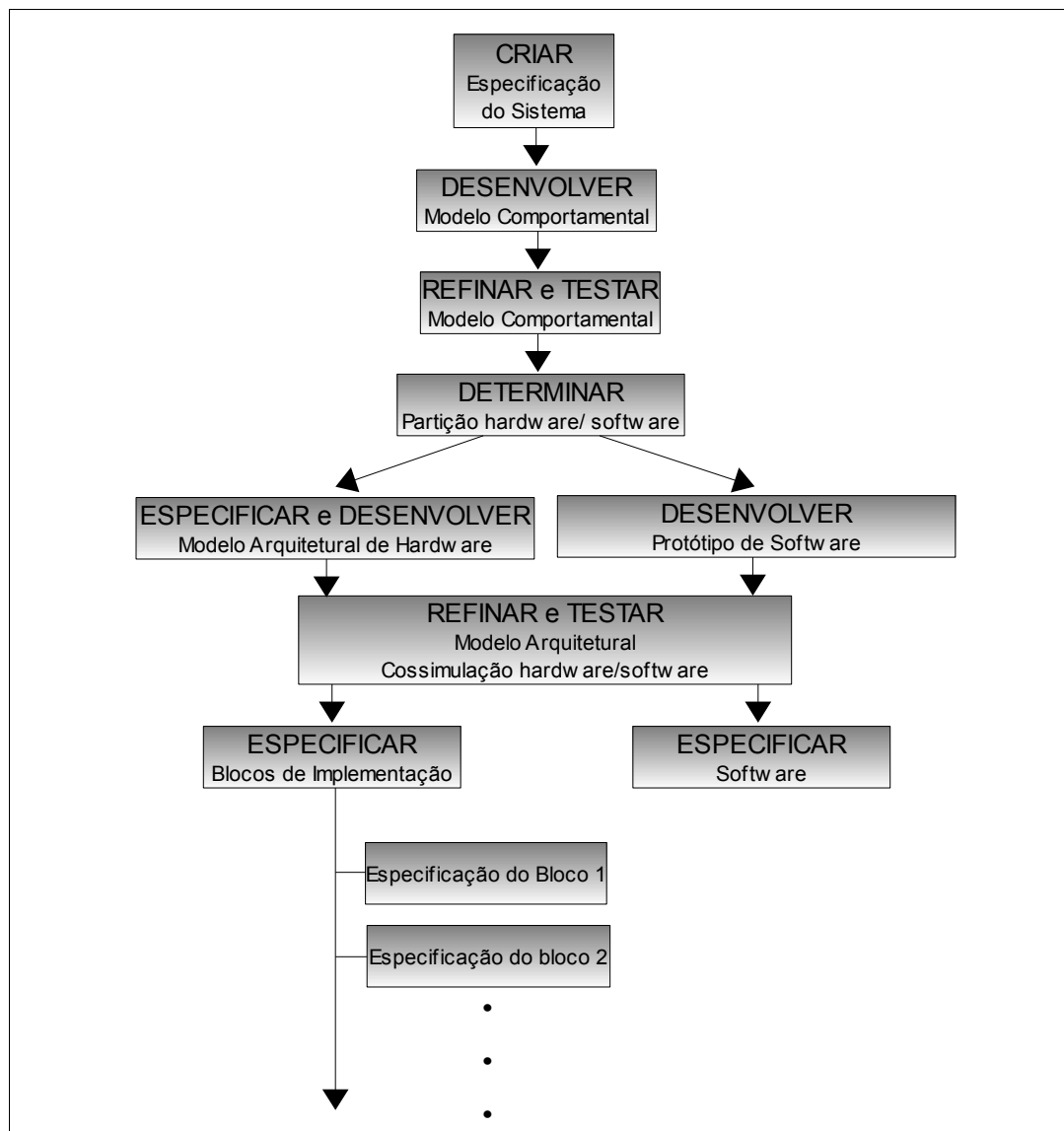


Figura 4: Modelo de processo de projeto de um sistema (KEATING, BRICAUD, 2002).

O processo de projeto de um sistema, como pode ser visualizado na figura 4, consiste em propôr um projeto de sistema candidato e então desenvolver uma série de modelos para avaliar e refinar o projeto. O processo de projeto a seguir, proposto por Keating e Bricaud (2002), envolve as seguintes etapas:

1. Criar a especificação do sistema;
2. Desenvolver um modelo comportamental;
3. Refinar e verificar o modelo comportamental;
4. Determinar, quando for o caso, a partição hardware/software (decomposição);

5. Especificar e desenvolver os modelos de arquitetura correspondentes;
6. Refinar e verificar o modelo arquitetural;
7. Especificar blocos a serem implementados.

2.1.4 Projeto no Nível de Sistema

Um **IP** bem projetado é a chave para o sucesso de um projeto SoC (KEATING, BRICAUD, 2002). Um conjunto de componentes IP deve estar disponível e ser aderente à plataforma definida, quando o projeto baseado em IP é combinado com uma abordagem baseada em plataforma (WAGNER, CESARIO, CARRO, JERRAYA, 2004).

Um IP precisa possuir ferramentas que suportem seu uso, além de funções e métodos usados bem definidos. Deve ser um produto pronto para usar (GAJSKI, WU, CHAIYAKUL, et al, 2000).

Um IP pode ser definido em termos de funcionalidade, estilo e tipo. O estilo pode ser de simples instância ou estilo parametrizável. O aumento de generalidade aumenta o valor de um IP, porém aumenta também a dificuldade de otimização, verificação e teste (GAJSKI, WU, CHAIYAKUL, et al, 2000).

Os IPs são normalmente classificados em *Hard IPs*, *Soft IPs* e *Firm IPs*. Um *Hard IP* é facilmente caracterizável (previsível) e dificilmente modificável (portável) (GAJSKI, WU, CHAIYAKUL, et al, 2000). É caracterizado por ter todas as portas e conexões posicionadas e roteadas (WAGNER, CESARIO, CARRO, JERRAYA, 2004). Um *Soft IP* é dificilmente caracterizável e facilmente modificável (GAJSKI, WU, CHAIYAKUL, et al, 2000). É caracterizado por ter apenas uma representação RTL (WAGNER, CESARIO, CARRO, JERRAYA, 2004). Um *Firm IP*, por sua vez, pode ser entregue na forma RTL ou de *netlist*, com ou sem posicionamento detalhado, mas com alguma forma de informação física de projeto suplementar ao RTL (KEATING, BRICAUD, 2002).

Além disso, um IP pode ser licenciável, quando se licencia o projeto e conjunto de ferramentas, onde o projeto de sistema e fabricação podem ficar com o cliente. Ou pode ser proprietário, onde o fornecedor protege sua propriedade intelectual (PALMA, MORAES, CALAZANS, 2001).

A tabela 1 apresenta um modo de classificar os IP cores proposto pelos mesmos autores.

Tabela 1: Três classificações de IP cores baseado em cinco critérios (PALMA, MORAES, CALAZANS, 2001).

	<i>Hard IP</i>	<i>Firm IP</i>	<i>Soft IP</i>
Rigidez	A organização é pré-definida.	Combinação de código fonte e <i>netlist</i> dependente de tecnologia.	Apresenta um código fonte comportamental independente de tecnologia.
Modelagem	Modelado como um elemento de biblioteca.	Combinação de blocos sintetizáveis fixos. Permite compartilhar recursos com outros cores.	Sintetizável com diversas tecnologias.
Flexibilidade	Não pode ser modificado pelo projetista.	A personalização de funções específicas é dependente de tecnologia.	O projeto pode ser modificado e independente da tecnologia.
Previsibilidade	Temporização é garantida.	Caminhos críticos com temporizações fixas.	A temporização não é garantida, podendo não atender os requisitos do projeto.
Proteção de propriedade intelectual	Alta. A descrição normalmente corresponde a um <i>leiaute</i> .	Média.	Muito pequena. Código fonte aberto.

O projeto de um IP pode ser dividido em fases de projeto. As diversas maneiras de se projetar um IP podem fazer essa divisão diferentemente uma das outras. Não existe um modo único de projetar um IP. O caminho proposto por Keating e Bricaud (2002) serviu de base para o trabalho dessa dissertação. Os parágrafos a seguir explicam esse modo de projetar o IP proposto.

Durante o projeto de um IP, o projetista deve levar em consideração algumas características (KEATING, BRICAUD, 2002). Entre elas, podemos citar:

- Configurável, para as necessidades de diversos projetos;
- Interfaces padrões, para possibilitar integração com outros componentes;
- Aderente às boas práticas de projeto e codificação, para facilitar a verificação e

- empacotamento para reuso;
- Conjunto completo de entregáveis, com RTL sintetizável, IP de verificação, *scripts* de síntese e documentação.

Para permitir que projetistas SoC usem o IP de implementação efetivamente, o desenvolvedor de IP deve fornecer um IP de verificação tão bom quanto o IP de implementação.

As cinco primeiras **fases de projeto** de um IP são:

- 1) Definição das Características Chaves;
- 2) Planejamento e Especificação;
- 3) Projeto e Verificação do IP de Implementação e IP de Verificação;
- 4) Empacotamento;
- 5) Teste Alfa e Lançamento de Versão.

As características chaves incluem a funcionalidade básica e parâmetros de configuração que irão permitir ao componente ser usado por várias aplicações alvo. É necessário um completo entendimento da aplicação para a definição das características chaves. Após a definição das características chaves, parte-se para o desenvolvimento de uma especificação funcional detalhada para o componente, uma especificação detalhada para o IP de verificação e um planejamento detalhado para o resto do projeto. O componente é então implementado e verificado detalhadamente. Um teste adicional é realizado e os entregáveis finais, incluindo a documentação de usuário, são empacotadas de forma a serem entregues ao usuário final. Os entregáveis finais são então testados para se ter certeza que estão completos e prontos para serem usados pelo projetista SoC, finalizando, assim, as primeiras fases de um IP (KEATING, BRICAUD, 2002).

2.1.5 Projeto no Nível de Arquitetura

Assume-se que, para o início do projeto no nível de arquitetura, um circuito é especificado por um conjunto de **recursos funcionais**, um conjunto de **restrições** e um **grafo de sequenciamento** (DE MICHELI, 1994).

Os **recursos** implementam diferentes tipos de função em hardware. Classificam-se como recursos funcionais, recursos de memória e recursos de interface. Os recursos funcionais podem ser primitivos, quando são projetados para funções lógicas padrões, ou específicos, quando resolvem sub-tarefas particulares. Os recursos de memória, por sua vez, armazenam dados, e os recursos de interface dão suporte à transferência de dados (DE MICHELI, 1994).

As **restrições** podem ser de interface ou de implementação. As restrições de interface relaciona-se ao formato e temporização das transferências de dados de E/S, enquanto que as restrições de implementação refletem o desejo do projetista de alcançar uma estrutura com algumas propriedades. Restrições de área e desempenho, representadas por limites de tempo de ciclo e de latência, são exemplos de restrições de implementação (DE MICHELI, 1994).

O **grafo de sequenciamento**, assim como o grafo de fluxo de dados, são modelos abstratos – baseados em grafos – usados para representar diferentes domínios nos níveis lógico e arquitetural. Modelos abstratos do domínio comportamental no nível de arquitetura são definidos em termos de **tarefas** (ou **operações**) e suas **dependências**. As tarefas podem ser *No-Operations* (NOPs), ou seja, falsas operações que são executadas instantaneamente sem nenhum efeito. E as dependências aparecem devido à razão da disponibilidade de dados para uma operação que precisa de outra feita anteriormente, e também por causa da forma serial das restrições na especificação (DE MICHELI, 1994).

Os modelos comportamentais no nível arquitetural são capturados por grafos de sequenciamento. A síntese consiste em dois estágios. Primeiramente, posiciona-se as operações no **tempo** e no **espaço**, determinando o intervalo de tempo para sua execução e seu compartilhamento de recursos. Após, determina-se o **detalhamento das interconexões** do caminho de dados e as especificações no nível lógico da unidade de controle (DE MICHELI, 1994).

A tarefa de determinar os tempos iniciais, referentes às restrições precedentes especificadas por um grafo de sequenciamento, se chama **scheduling**. Tempo inicial de uma operação é definido como o tempo pelo qual a operação inicia sua execução (DE MICHELI, 1994).

Por sua vez, um conceito fundamental, que relaciona operações e recursos, é o

binding, que especifica que recurso implementa uma operação. Um caso simples de **binding** é um recurso dedicado, onde cada operação relaciona-se a um recurso, em uma função de um para um (DE MICHELI, 1994).

O detalhamento das interconexões, ou síntese de conectividade, envolve a completa definição da visão estrutural do caminho de dados, ou seja, refinar a informação de **binding** presente na especificação de todas interconexões. De forma prática, consiste em definir as interconexões entre recursos, circuitos de *steering logic* (multiplexadores ou barramentos), recursos de memória (registradores e matrizes de memória), portas de entrada/saída e unidade de controle. Diferentes estilos podem ser usados, como caminho de dados baseados em barramentos, macro-células ou matriz (DE MICHELI, 1994).

Em relação à unidade de controle, os recursos sequenciais necessitam de um sinal de *start* – e algumas vezes de um *reset*. Consequentemente, a execução de cada operação necessita de um conjunto de sinais de **ativação**, que precisa ser enviado pela unidade de controle. Para completar, a unidade de controle recebe alguns sinais condicionais necessários para avaliar algum tipo de construção. Os sinais condicionais fornecidos por operações dependente de dados são chamados de sinais de **conclusão**. O conjunto desses pontos de controle precisa ser identificado (DE MICHELI, 1994).

2.1.6 Projeto no Nível de Microarquitetura

A microarquitetura pode ser vista como a **conexão entre lógica e arquitetura**. A microarquitetura é organização específica de registradores, ULAs, máquinas de estados finitas (FSMs), memórias e outros blocos de construção lógicos necessários para implementar uma arquitetura. Uma arquitetura pode possuir diferentes microarquiteturas com diferentes compromissos entre desempenho, custo e complexidade. Todas realizam as mesmas operações, mas seu projeto interno pode variar muito (HARRIS, HARRIS, 2007).

Nesse nível de abstração, define-se como cada bloco será implementado, as partições internas dos blocos, assim como o tamanho e quantidade de memória presentes, ou seja, define-se mecanismos e estruturas específicas para se alcançar a implementação. A implementação pode ser representada textualmente, graficamente ou em forma de software.

2.1.7 Projeto no Nível RTL

O nível de transferência de registrador (*RTL – register transfer level*) é um meio de descrever a operação de um circuito digital onde o comportamento é definido em termos de um **fluxo de sinais** entre registradores e de **operações** realizadas.

Na eletrônica digital, um circuito pode ser definido como uma rede que processa variáveis de valores discretos. Um circuito pode ser visto como uma caixa-preta com um ou mais terminais de valores discretos de entrada, um ou mais terminais de valores discretos de saída, uma especificação funcional descrevendo o relacionamento entre as entradas e as saídas, e uma especificação de temporização descrevendo o atraso entre a carga da entrada e a resposta da saída (HARRIS, HARRIS, 2007).

Dentro da caixa-preta, o circuito é composto por **nós e elementos**. Um elemento é também um circuito com entradas, saídas, e uma especificação. Um nó é um fio no qual a tensão conduz uma variável com valor discreto. Os nós são classificados como de entrada, de saída e interno (HARRIS, HARRIS, 2007).

Os circuitos digitais são classificados como **combinacional** ou **sequencial**. Uma saída de um circuito combinacional depende apenas dos atuais valores de entrada para computar sua saída. Um circuito sequencial depende de uma sequência de entrada. Um circuito combinacional não possui memória, mas um circuito sequencial possui memória (HARRIS, HARRIS, 2007). O RTL de um circuito sequencial define os detalhes de implementação em termos de elementos de armazenamento sequencial, como registradores e memórias.

A lógica sequencial deve explicitamente lembrar certas entradas anteriores, ou organizar essas entradas em um conjunto de informações chamadas de **estado** de um sistema. O estado de um circuito digital sequencial é um conjunto de bits chamadas de variáveis estado que contém toda informação sobre o passado, necessário para explicar o comportamento futuro de um circuito (HARRIS, HARRIS, 2007).

Circuitos sequenciais síncronos podem ser desenhados na forma de máquinas de estados finitas (FSMs – *Finite State Machines*). Uma FSM consiste em dois blocos de lógica combinacional, *next state logic* e *output logic*, e um registrador que armazena o estado. A cada borda de relógio, a FSM avança para o próximo estado, que é computado baseado no estado atual e entradas. Existem duas classes de máquinas de estados

finitas, caracterizado por suas especificações funcionais. Nas máquinas de Moore, as saídas dependem apenas do estado atual da máquina. Nas máquinas Mealy, as saídas dependem do estado atual e das entradas atuais. As máquinas de estados finitas proporcionam um caminho sistemático de projetar circuitos sequencias síncronos dada uma especificação funcional (HARRIS, HARRIS, 2007).

O processo de encontrar um conjunto eficiente de portas lógicas para realizar uma dada função é um trabalho intensivo e propenso a erros. Os projetistas descobriram que é muito mais produtivo trabalhar em um alto nível de abstração, especificando apenas as funções lógicas e permitir que uma ferramenta CAD produza portas otimizadas. As especificações nesse nível são geralmente dadas em forma de HDL (HARRIS, HARRIS, 2007).

As duas linguagens mais conhecidas são Verilog e VHDL, ambas são capazes de descrever qualquer sistema de hardware e possuem suas peculiaridades. Comparado a Verilog, o VHDL é mais prolixo e embaraçoso. O VHDL teve como primeira previsão de uso a documentação, embora também tenha sido adotada rapidamente por ferramentas de simulação e de síntese (HARRIS, HARRIS, 2007).

2.1.8 Estratégias de Verificação

Atribui-se à verificação funcional as **tarefas** de demonstrar que o projeto está correto, de tentar encontrar erros de projeto e de demonstrar que o que se pretende com o projeto – especificação – é preservado em sua implementação.

O principal **propósito** de uma verificação funcional é assegurar que o projeto implementa a funcionalidade pretendida. Uma das abordagens de verificação funcional é a verificação caixa-preta, onde não se pode visualizar ou saber sobre o interior de um projeto. Nessa abordagem, os casos de testes são independentes de implementação e podem ser usados em todos os níveis de projeto (BERGERON, 2006).

Existem diferentes níveis de verificação (BERGERON, 2006): no nível de unidade, de bloco, de núcleo, de sistema e de placa.

No nível de especificação de alto-nível, o processo de verificação foca no comportamento e não na implementação. Modelar construções em alto-nível requer menos de 10% do tempo necessário para modelar construções sintetizáveis. E não é

apenas menos código para escrever. Especificar e verificar em alto-nível, é mais simples e requer menor esforço para assegurar que o modelo está correto, além de possuir o benefício de melhorar o desempenho da simulação (BERGERON, 2006).

A figura 5 apresenta as fases de verificação em um fluxo de projeto.

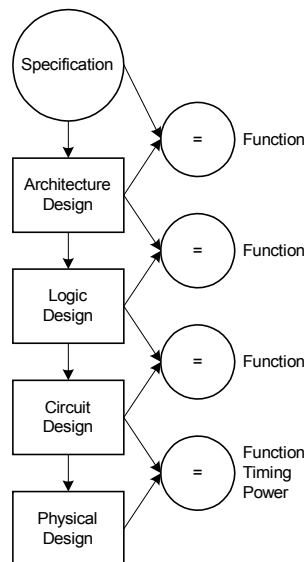


Figura 5: Fluxo de projeto e de verificação (WEST, HARRIS, 2005).

Uma variedade de ferramentas e técnicas são usadas para realizar a verificação funcional, como: modelos funcionais de barramento (BFM – *Bus Functional Model*), monitores para conferir respostas, linguagens de verificação de alto-nível, testes de regressão automatizados, ferramentas de cobertura de código, e outras. Componentes do ambiente de verificação, como modelos funcionais de barramentos e monitores, devem ser entregues junto com o componente IP para facilitar a verificação no nível de chip (KEATING, BRICAUD, 2002).

A verificação de um componente consiste em três fases principais: a verificação de sub-blocos individuais, a verificação do componente e a prototipagem. De forma geral, os erros são mais facilmente encontrados e reparados no nível de sub-blocos do que no nível de componente IP. A **observabilidade** e a **controlabilidade** são tipicamente mais fáceis em pequenos trechos de código. Uma verificação a nível de sub-blocos bem feita pode acelerar toda a verificação (KEATING, BRICAUD, 2002).

Em cada fase do processo de verificação, existe a necessidade de decidir quais

tipos de testes e quais ferramentas serão usadas. Os **tipos básicos de testes** de verificação incluem o teste de conformidade, teste de casos complexos, teste aleatório, teste do próprio código, teste de regressão, conferência de propriedades. As **ferramentas** disponíveis incluem simulação, linguagens de verificação de hardware, IP de verificação, ferramentas de cobertura de código, modelagem de hardware, emulação, prototipagem. A inspeção também pode ser usada como forma de verificação. A inspeção de projeto e de código é considerada o meio mais rápido, barato e eficaz de se detectar e remover erros (KEATING, BRICAUD, 2002).

Uma das técnicas de verificar um projeto é analisar a **taxa de cobertura** da verificação. A cobertura se trata de uma porcentagem da característica verificada, medindo o que realmente está sendo verificado pelo ambiente de verificação do total possível para essa característica. Existem métricas de cobertura funcional e de código. Essas métricas de cobertura fornecem a melhor indicação sobre a verificabilidade de um projeto, permitindo especificar medidas de quais funcionalidades e correspondentes níveis hierárquicos estão sendo verificadas (KEATING, BRICAUD, 2002).

Como parte de toda estratégia de verificação, uma análise estática de tempo deve ser realizada nas *netlists* resultantes da síntese para verificar se os objetivos de temporização do componente IP são alcançados. A verificação estática de tempo é o método mais eficiente de verificar o desempenho em tempo do componente IP. A simulação a nível de portas tem um uso limitado na verificação de temporização, sendo mais útil a sua utilização para lógica assíncrona (KEATING, BRICAUD, 2002).

2.1.9 Medidas de Qualidade

Para fins de avaliação, considera-se a **área** e o **desempenho** como medidas de qualidade.

A área de um circuito é medida pela soma das áreas de seus componentes. Os componentes fundamentais de circuitos digitais são as **portas lógicas** e **registradores**, que possuem áreas conhecidas *a priori*. A área das interconexões também desempenha um papel importante, e não deve ser negligenciada. Essa área pode ser derivada a partir de uma visão física completa, ou estimada a partir de uma visão estrutural usando modelos estatísticos (DE MICHELI, 1994).

A computação do desempenho de um circuito, por sua vez, necessita da análise de sua estrutura e, frequentemente, de seu comportamento. O significado de desempenho varia de acordo com as diferentes classes de circuitos digitais. Para circuitos lógicos combinacionais, o desempenho é medido pelo **atraso de propagação** entre entrada e saída. Para circuitos sequenciais síncronos, o desempenho é medido por (DE MICHELI, 1994):

- **Tempo de ciclo** ou período de relógio. É o período do relógio mais rápido que pode ser aplicado ao circuito;
- **Latência** ou atraso de execução. É o tempo necessário para executar operações. A medida pode ser feita em termos de ciclos de relógio;
- **Tempo de execução**. É o produto do tempo de ciclo com a latência.

Existe ainda uma medida adicional de desempenho para circuitos *pipeline*, o *throughput* (ou taxa de computação), que é uma taxa pelo qual os dados são produzidos e consumidos (DE MICHELI, 1994).

2.2 Sistema Brasileiro de TV Digital – SBTVD

Vários serviços surgiram a partir da evolução das tecnologias baseadas na digitalização da informação. Um destes casos é a televisão digital. Uma vantagem de grande importância em favor do surgimento da televisão digital é a possibilidade de difundir qualquer tipo de informação digital, além do áudio e do vídeo. Ao longo dos anos, foram definidos sistemas abertos de televisão digital. No caso do Brasil, definiu-se como padrão o Sistema Brasileiro de Televisão Digital – SBTVD.



Figura 6: Arquitetura do Sistema Brasileiro de Televisão Digital.

Um sistema de televisão pode ser dividido, basicamente, em três componentes: produtor de conteúdo, transmissão e recepção. O produtor de conteúdo é responsável pela produção e armazenamento do conteúdo, o sistema de transmissão é o processo de difusão (radiodifusão, cabo, satélite, etc.) de programas ou serviços, para transmissão de informações entre emissora e usuários, e o sistema de recepção é a parte que fica no usuário e recebe as informações da emissora.

O **processo de geração** do sinal de TV digital é realizado através codificação e compressão da informação a ser enviada. Dá-se o nome de fluxo elementar (ES – *Elementary Stream*) ao fluxo de dados da saída do codificador. O relacionamento de fluxos elementares formam o conceito de serviço. Os serviços são multiplexados em uma sequência de dados chamada de fluxo de transporte (TS – *Transport Stream*).

O **processo de recepção** é realizado através da captação do sinal difundido, demodulação para restauração do fluxo de transporte, demultiplexação para extração dos fluxos elementares de determinado serviço, decodificação de áudio e vídeo e, possivelmente, processamento, execução e exibição de outros dados.

O terminal de acesso faz parte do sistema de recepção e possui o objetivo de receber o sinal digital de televisão e exibi-lo ao usuário. Sua principal característica é a capacidade de processamento.

2.2.1 Arquitetura Básica de um Terminal de Acesso

A configuração básica do sistema de recepção é composto por antena, terminal de acesso e televisão (ABNT NBR 15604, 2007). A arquitetura do **terminal de acesso** é bem semelhante à de um computador. A norma ABNT NBR 15604 especifica o conjunto de funcionalidades essenciais necessárias aos dispositivos de recepção de televisão digital. Esses requisitos são apresentados na figura 7.

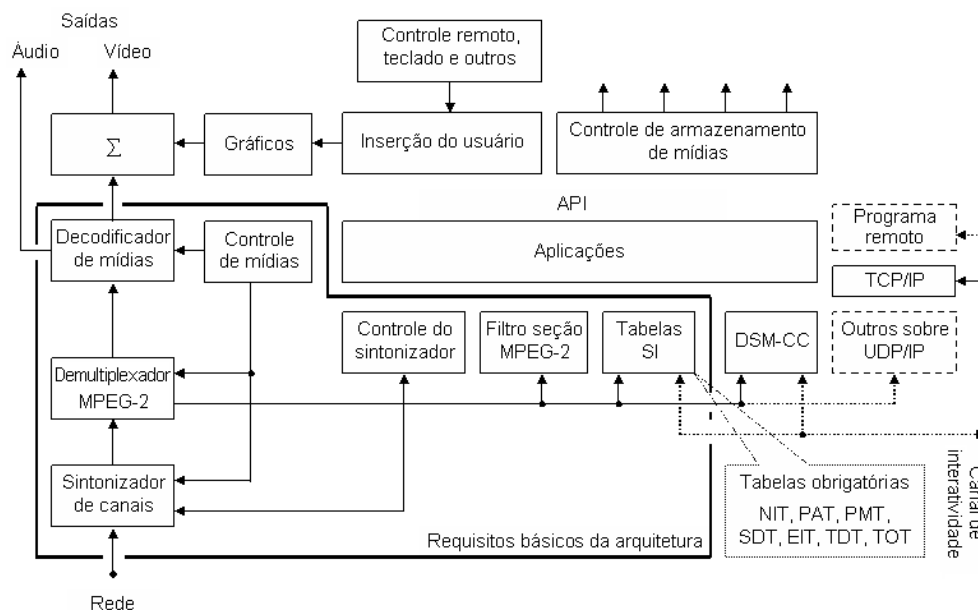


Figura 7: Requisitos básicos da arquitetura do receptor (ABNT NBR 15604, 2007).

Os terminais de acesso devem possuir características para atenderem seu objetivo básico, que é receber o sinal digital de televisão e exibi-lo. No terminal devem estar presentes módulos de recepção, decodificação de sistema, decodificação de áudio, decodificação de vídeo, processamento de sinais, núcleo de processamento e interfaces de comando. A seguir, é apresentada a função de cada módulo da arquitetura de referência apresentada na figura 8:

- Os módulos de recepção (Sintonizador e Demodulador), ou *Front end*, são responsáveis pela recepção, sintonia e demodulação do sinal recebido pela transmissora.
- O módulo de decodificação MPEG-2 Sistema (Demultiplexador de Transporte) é

responsável pela decodificação de sistema – camada de transporte.

- O módulo de decodificação de áudio (Decodificador de Áudio) pela decodificação do áudio.
- O módulo de decodificação de vídeo (Decodificador de Vídeo) pela decodificação do vídeo.
- O módulo de sincronização (Sincronizador de Áudio e Vídeo) é responsável pelas informações de sincronização do áudio e vídeo para apresentação/decodificação.
- O módulo de recuperação de relógio é responsável por recuperar o relógio do sistema no terminal de acesso.
- O módulos de processamento de vídeo (Processamento de Visualização e Gráficos) é responsável pelo tratamento do sinal digital de vídeo para apresentação.
- O núcleo de processamento é responsável por processar os comandos de usuário recebidos via interface de usuário, controlar o sintonizador e configurar os dispositivos de conversão e codificação dos sinais de áudio e vídeo.
- O módulo de interface de comando (Informação do Sistema/Navegação) é responsável por permitir a interação do usuário com o terminal.

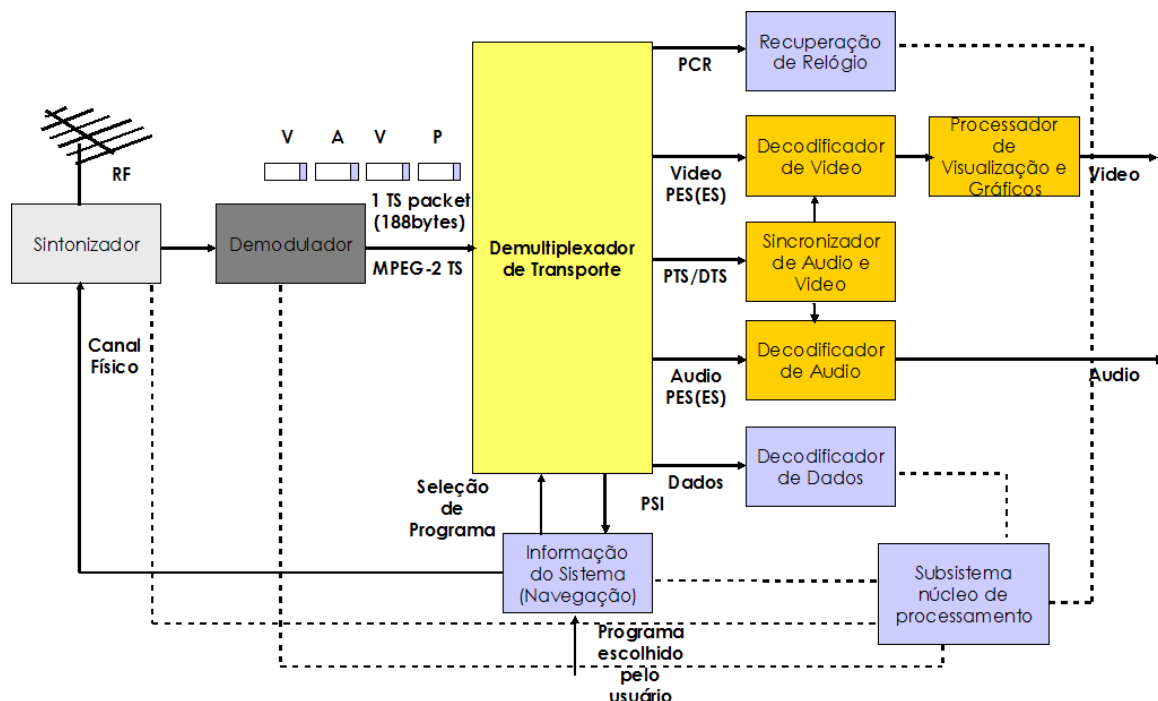


Figura 8: Diagrama de blocos geral da arquitetura de referência para o receptor.

2.2.2 Decodificação do fluxo de entrada

O fluxo de transporte recebido para apresentação, contém uma primeira **codificação em nível sistema**, que é o encapsulamento dos diversos fluxos (áudio, vídeo, programas, informações de conteúdos, etc) dentre os conteúdos há outros níveis de codificação que servirão a compressão de sinais de áudio, vídeo, dados encapsulamento de aplicativos, interatividade, etc. O escopo deste estudo foi o decodificador de sistema traduzido também como demultiplexador de transporte, no caso do SBTVD o MPEG-2 TS.

O **demultiplexador de transporte** tem a **função** de extrair os pacotes de áudio, vídeo, sincronização dos pacotes, dados de recuperação de relógio, dados privados e serviços de navegação. Ele poderá enviar esses dados para um ambiente de memória compartilhada, ou diretamente para outros componentes do receptor digital.

2.2.3 Demultiplexador de MPEG-2 Sistema – MPEG-2 TS

Esse bloco funcional realiza a análise gramatical do fluxo de transporte para extrair os fluxos elementares empacotados para entregá-los aos decodificadores de áudio e vídeo. Ele deve prover um **conjunto completo de funções** de demultiplexação, incluindo sincronização, filtragem de PID, recuperação de relógio, filtragem de tabelas de seção, verificação de CRC, processamento de dados e transferência, tratamento de erros e interfaces externas.

O decodificador de sistema filtra fluxos elementares de áudio e vídeo, e de seções de tabela extraídos do TS, e os envia para o decodificador de vídeo e para o decodificador de áudio.

A demultiplexação é necessária para reconstituir fluxos elementares a partir do TS. O código de pacote ID no TS torna isso possível (ISO/IEC 13818-1, 2000).

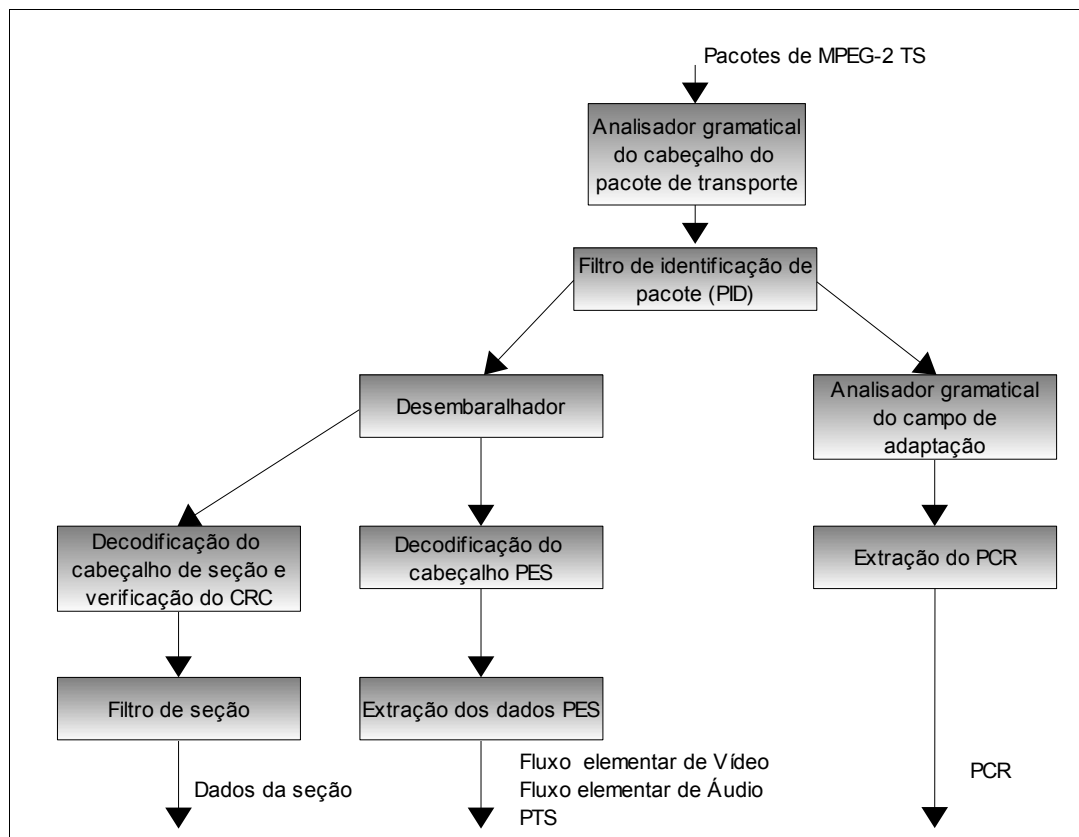


Figura 9: Fluxo de dados através do demultiplexador.

2.2.4 O Padrão MPEG-2 Sistema

O MPEG-2 é um conjunto de nove especificações. O MPEG-2 Sistema (ISO/IEC 13818-1, 2000) foi adotado como o padrão para o SBTVD e é adotado amplamente no nível internacional e especifica a estrutura multiplexada para combinação de dados de áudio e vídeo, além de dar um significado para a representação da informação de tempo necessário para reprodução de sequências sincronizadas em tempo real (HANNA, GILLIES, COCHON, et al, 1995).

O padrão MPEG-2 Sistema especifica a **camada de sistema** da codificação e define dois níveis de protocolo: o **fluxo de transporte** usado em ambientes onde erros podem ocorrer, e o **fluxo de programa** usado em ambientes livre de erros (INAMORI, NAGANUMA, WAKABAYASHI, ENDO, 1996).

A abordagem básica de multiplexação para os fluxos elementares de áudio e vídeo

é ilustrado na Figura 10.

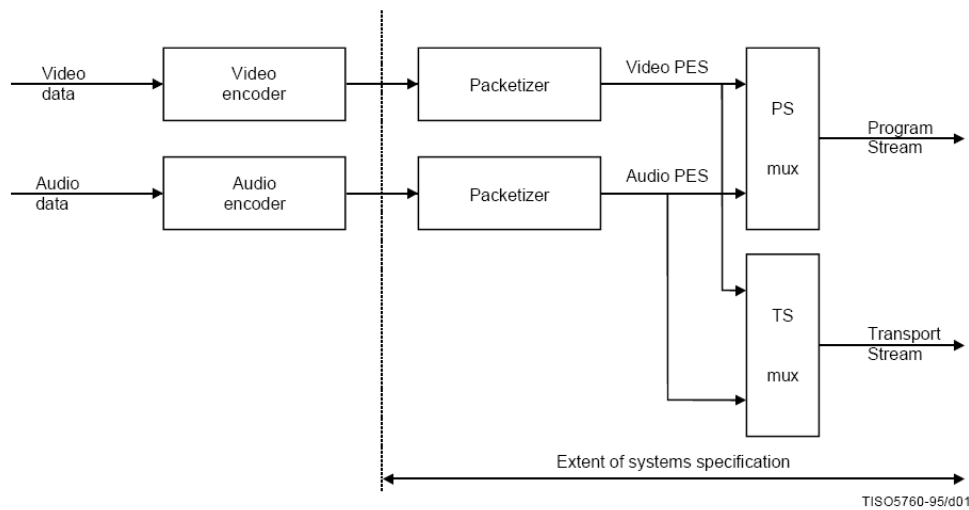


Figura 10: Abordagem básica de multiplexação para fluxos elementares de áudio e vídeo do padrão MPEG-2 Sistema (ISO/IEC 13818-1, 2000).

Um **esquema de empacotamento hierárquico** é usado no protocolo MPEG-2 Sistema. Primeiramente, o fluxo elementar do codificador é empacotado em um **PES** (*Packetized Elementary Stream*). Depois, o PES é empacotado em um **TS** ou **PS** dependendo da aplicação. Este esquema é apresentado nas figuras 11 e 12. Alguns valores do cabeçalho PES são determinados através da análise do fluxo elementar. A **marca de tempo de apresentação** (PTS – *Presentation Time Stamp*) e a **marca de tempo de decodificação** (DTS – *Decoding Time Stamp*), usadas para sincronizar o vídeo com o áudio, são exemplos desses valores. A sintaxe do TS/PS depende do fluxo elementar, o que complica o processo de empacotamento (INAMORI, NAGANUMA, WAKABAYASHI, ENDO, 1996).

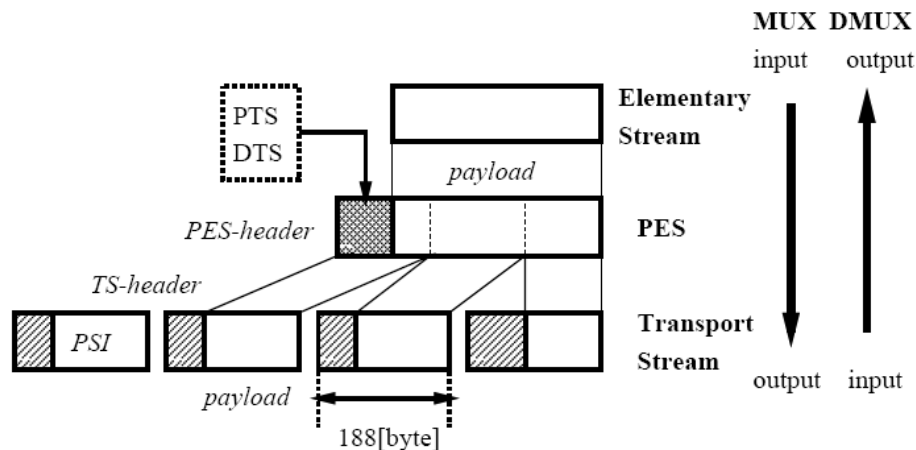


Figura 11: Composição do fluxo de transporte – TS (INAMORI, NAGANUMA, WAKABAYASHI, ENDO, 1996).

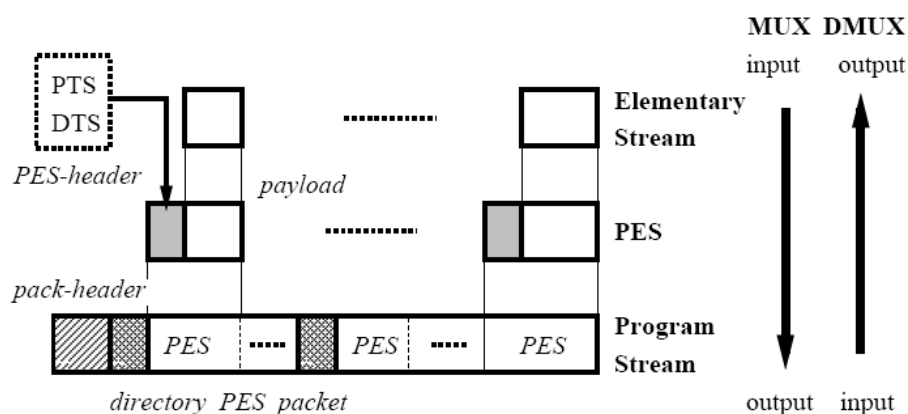


Figura 12: Composição do fluxo de programa – PS (INAMORI, NAGANUMA, WAKABAYASHI, ENDO, 1996).

2.2.5 Visão Geral do Pacote TS

O TS é construído em duas camadas: **camada de sistema** e **camada de compressão**. O fluxo de entrada do demultiplexador de TS possui a camada de sistema sobre uma camada de compressão. Os fluxos de entrada dos decodificadores de áudio e vídeo possuem apenas a camada de compressão (ISO/IEC 13818-1, 2000).

A **camada de sistema** do TS é dividida em duas sub-camadas, uma para operações a nível de multiplex – camada de **empacotamento TS** – e uma para

operações de fluxo individual – camada de **empacotamento PES**. Estão incluídas nas operações no nível de multiplex, a ordenação dos dados recuperados de um canal, o ajuste dos relógios e o gerenciamento de *buffers* (ISO/IEC 13818-1, 2000).

Os fluxos são temporariamente subdivididos em pacotes, e os pacotes são serializados. Os pacotes TS possuem 188 bytes de comprimento. Um pacote PES contém bytes codificados de um, e apenas um, fluxo elementar, e o tamanho do pacote pode ser fixo ou variável (ISO/IEC 13818-1, 2000).

A camada de empacotamento PES é independente da camada de compressão em alguns casos, mas não em todos. É independente, no sentido em que a *payload* dos pacotes PES não precisam iniciar nos códigos iniciais da camada de compressão (ISO/IEC 13818-1, 2000).

As regras sintáticas são aplicadas somente na camada de sistema, e não se estendem para a codificação das especificações de áudio e vídeo da camada de compressão, entretanto as regras semânticas são aplicadas ao fluxo combinado por completo (ISO/IEC 13818-1, 2000).

O trabalho do demultiplexador de fluxo de transporte é realizar a análise gramatical dessas estrutura de modo a reconstruir os fluxos elementares de programa. Um **fluxo elementar** consiste em todos os dados necessários para reconstruir uma parte específica do programa. As partes do programa normalmente consistem em áudio, vídeo, dados e informações específicas de programa (PSI – *Program Specific Information*). Para reconstruir o programa, o demultiplexador deve fornecer sincronização de pacotes, detecção de erros, identificação de programa, desembaralhamento e recuperação de relógio (BRIDGEWATER, DEISS, 1995).

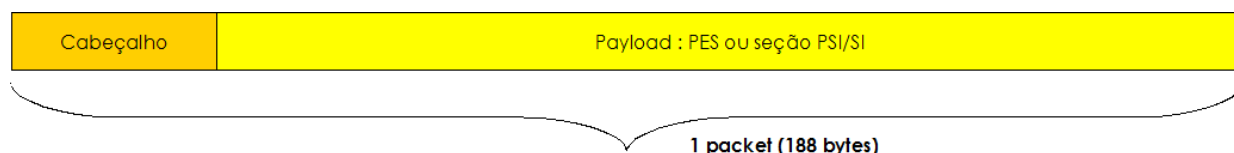


Figura 13: Pacote de Fluxo de Transporte.

stamp) que indica o tempo quando uma unidade de acesso de áudio ou vídeo é para ser decodificada; e a referência de relógio de fluxo elementar (*ESCR – elementary stream clock reference*) (ZHANG, et al, 2005).

2.2.7 Infraestrutura PSI/SI

Informação PSI é o dado que identifica que partes do fluxo TS pertence a um programa particular. De acordo com a norma brasileira ABNT NBR 15603-2 (2007), são usadas três tabelas PSI/MPEG-2 no SBTVD:

- PAT – A Tabela de Associação de Programa (*PAT – Program Association Table*), o qual possui valor de identificador de pacote (*PID – Packet Identifier*) sempre 0x0000, é o ponto de entrada para as tabelas PSI. Ela lista todos os programas no fluxo TS e associa cada programa com outro PID, que se refere a um pacote com uma Tabela de Mapa de Programa (*PMT – Program Map Table*) em sua *payload*.
- PMT – A tabela PMT lista todos os PIDs para os pacotes contendo elementos de um programa particular como áudio, vídeo, e referência de relógio de programa (*PCR – Program Clock Reference*), etc.
- CAT – A Tabela de Acesso Condicional (*CAT – Conditional Access Table*), o qual valor de PID é pré-definido como 0x0001, fornece a associação entre um ou mais sistema de acesso condicional (*CAS – Conditional Access System*). Ele contém PIDs para Mensagens de Gerenciamento de Autorização (*EMM – Entitlement Management Message*), que contém informação do nível de autorização para o CAS.

A mesma norma brasileira ABNT NBR 15603-2 (2007), usa quinze tabelas SI para a construção das **informações básicas**:

- BAT – A Tabela de Associação de Buquê (*BAT – Bouquet Association Table*) fornece informações sobre os buquês existentes e os serviços inclusos em cada buquê.
 - NIT – A Tabela de Informação de Rede (*NIT – Network Information Table*), o qual
-

valor de PID também está contido na tabela PAT, mapeia frequências de canais, números de transponder, e outras informações guia para programas.

- SDT – A Tabela de Descrição de Serviços (*SDT – Service Description Table*) informa serviços existentes em um TS.
 - EIT – A Tabela de Informação de Eventos (*EIT – Event Information Table*) fornece informações em ordem cronológica sobre os eventos existentes por serviço.
 - TDT – A Tabela de Data e Horário (*TDT – Time Date Table*) é utilizada como referência para informar data e hora do sistema.
 - TOT – A Tabela de Referência de Data e Horário (*TOT – Time Offset Table*) é responsável por informar ao receptor a hora, data, fuso horário e a existência de horário de verão.
 - RST – A Tabela de Estado do Evento (*RST – Running Status Table*) permite atualização rápida e precisa do estado de um ou mais eventos, como “*pausing*” ou “*running*”. É necessária quando ocorrem alterações de horário de programação.
 - ST – A Tabela de Preenchimento (*ST – Stuffing Table*) é utilizada para invalidar outras tabelas
 - PCAT – A Tabela de Anúncio de Conteúdo Parcial (*PCAT – Partial Content Announcement Table*) fornece um conteúdo parcial incluso na radiodifusão de dados.
 - BIT – A Tabela de Informação do Radiodifusor (*BIT – Broadcaster Information Table*) designa as unidades radiodifusoras e os parâmetros de serviço de informação (SI) para cada unidade radiodifusora existente.
 - NBIT – A Tabela de Informação de Grupo da Rede (*NBIT – Network Board Information Table*) transmite a informação de grupo de rede e a informação de referência para obtenção de grupo de rede.
 - LDT – A Tabela de Referência de Outras Tabelas (*LDT – Linked Description Table*) transmite informações sobre referência a outras tabelas.
 - LIT – A Tabela de Informação de Evento Local (*LIT – Local Event Information Table*) informa as instruções relacionadas a eventos locais, tais como discriminação por hora, nome e explicação sobre o evento em si (tipo de cenário etc.).
 - ERT – A Tabela de Relação de Eventos (*ERT – Event Relation Table*) indica as relações entre programas ou eventos locais, assim como grupos e atributos dos
-

programas e eventos locais.

- ITT – A Tabela de Transmissão de Índice (*ITT – Index Transmission Table*) descreve informações relacionadas aos índices dos programas, quando os programas devem obrigatoriamente ser transmitidos.

2.3 Evolução de Chips Comerciais MPEG-2 TS

A seguir, são apresentados chips comerciais que envolvem a demultiplexação de TS. Eles são classificados como: soluções integradas em um único chip (*single chip*), que englobam funcionalidades além da demultiplexação do fluxo MPEG-2 TS; soluções baseada em dois chips (*chipset*), com funcionalidades além da demultiplexação, com interfaces para controle por processador externo; e como soluções baseadas em chips isolados, que desempenham a tarefa específica de de-multiplexagem. A ordenação está na ordem cronológica.

2.3.1 SAA7205H (1997) – Philips

O chip isolado SAA7205H da Philips é um demultiplexador que possui uma arquitetura que inclui as seguintes entidades funcionais, como apresentado na figura 15: analisador sintático de MPEG-2, tratador de erros, filtro de *teletext*, filtro de dados genéricos, filtro de dados de alta velocidade, filtro de dados de vídeo, filtro de dados de áudio, processador de referência de relógio de programa, processador de marcas de tempo, *buffers* do tipo fila e interface de microcontrolador.

O analisador sintático de MPEG-2 faz a análise sintática do fluxo de transporte de acordo com a especificação de sistema MPEG-2. O tratador de erro é invocado quando um erro é detectado. O filtro de *teletext* filtra o fluxo de transporte com base no PID e o fluxo elementar de dados é armazenado em um *buffer* do tipo fila. Um filtro de dados genéricos é conectado à interface genérica. O filtro de fato não filtra, mas passa o fluxo de transporte em formato de byte. O filtro de alta velocidade recupera pacotes inteiros de transporte, de *payload* TS, de *payload* PES ou seções de um fluxo de entrada com base em um filtro programável. Um filtro de dados de vídeo seleciona dados PES ou fluxos elementares de dados (programáveis) com base em um PID programável, e passa para a

2.3.2 IBM39MPEGCS24PFA16C e IBM39MPEGCS24DPFA16C (1999) – IBM

Os chipsets IBM39MPEGCS24PFA16C e IBM39MPEGCS24DPFA16C da IBM possuem uma arquitetura que consiste em seis blocos funcionais, como apresentado na figura 16: demultiplexador de transporte, decodificador de vídeo, dois decodificadores de áudio, interface para CPU, interface de apresentação e interface de memória.

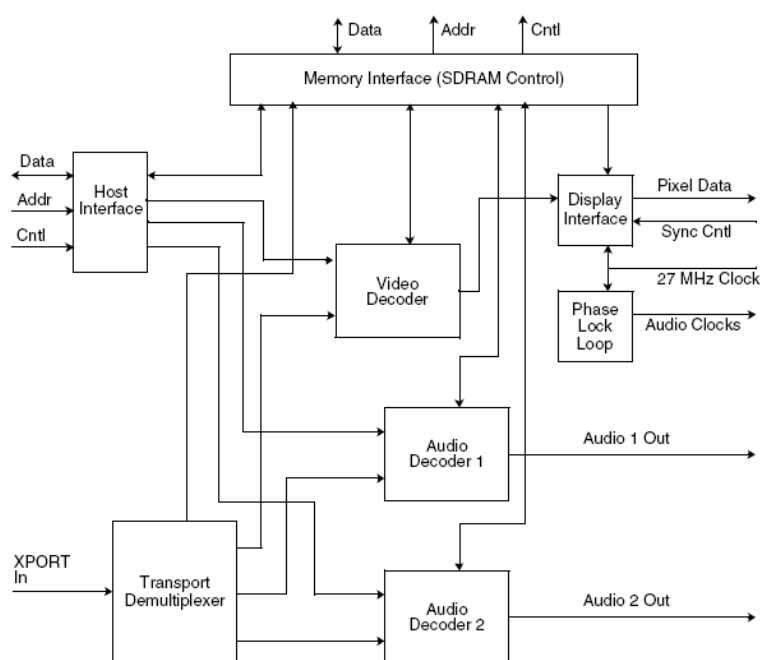


Figura 16: Diagrama de blocos do IBM39MPEGCS24PFA16C e IBM39MPEGCS24DPFA16C (1999).

O demultiplexador de transporte suporta taxa máxima de entrada de 60 Mbps (serial) ou 160 Mbps (paralelo), possui filtragem de seleção de tabela e filtragem de identificação de pacote. Esse bloco funcional realiza a análise gramatical do fluxo de transporte MPEG-2 para extrair fluxos elementares empacotados para decodificadores de áudio e vídeo. Ele provê um conjunto completo de funções de demultiplexação, incluindo sincronização, filtragem de PID, recuperação de relógio, filtragem de tabelas de seção, verificação de CRC, processamento de dados e transferência, tratamento de erros e interfaces externas. Essas funções básicas são configuradas pela aplicação.

A sincronização é feita através da detecção do *byte* de sincronização e

estabelecimento dos limites do pacote de transporte.

Para filtrar o fluxo de transporte são usados até 32 valores de PID. O filtro de PID associa um índice de PID de 5 bits para cada uma das 32 entradas. O índice de PID 31 é reservado para o PID de vídeo, o índice de PID 30 é reservado para o PID de audio1 e o índice de PID 29 é reservado para o PID de audio2. Os outros são definidos pela aplicação.

O demultiplexador de transporte auxilia na recuperação de relógio de programa em um fluxo de transporte extraíndo os PCRs de um PID indicado, calculando o *offset* a partir do valor do relógio de tempo do sistema (STC - *System Time Clock*) e compara ele a um limiar definido pela aplicação para determinar se uma correção na frequência de relógio é necessária.

Quando uma mudança na base de tempo do sistema ocorre no fluxo de PID do PCR, o demultiplexador de transporte carrega automaticamente o STC com o novo valor.

O demultiplexador de transporte fornece um conjunto básico de funções de processamento de dados, que não requer suporte de processador para manipulação de pacotes individuais.

O hardware de transporte analisa gramaticalmente os campos do cabeçalho de transporte e de adaptação como parte das funções do filtro de PID, de recuperação de relógio e tratamento de erros. Pode filtrar as seções de tabela, ser programado para verificar o valor de CRC das seções de tabela em relação ao codificado no fluxo.

O demultiplexador de transporte pode transferir a totalidade ou parte de qualquer pacote para a memória do sistema usando 32 filas independentes (uma para cada índice de PID). Simplifica as tarefas de tratamento de erros a nível de sistema e de mudanças na base de tempo usando a comunicação de dados comprimidos para se comunicar diretamente com os decodificadores.

A figura 17 apresenta o fluxo de dados do fluxo de transporte através do módulo de transporte. O bloco de sincronização encontra o início do pacote de transporte. O analisador gramatical de pacote extrai os dados do cabeçalho do pacote de transporte e do campo de adaptação. O PID é um desses campos, que é comparado com os PIDs ativos no filtro de PID. Se o PID combinar com um dos valores pré-definidos, os campos restantes são extraídos e o pacote é enviado para o carregador de pacote. Concorrentemente, o analisador gramatical de pacote envia PCRs dos pacotes de PCR

que combinam para a unidade de recuperação de relógio para reconstrução do STC.

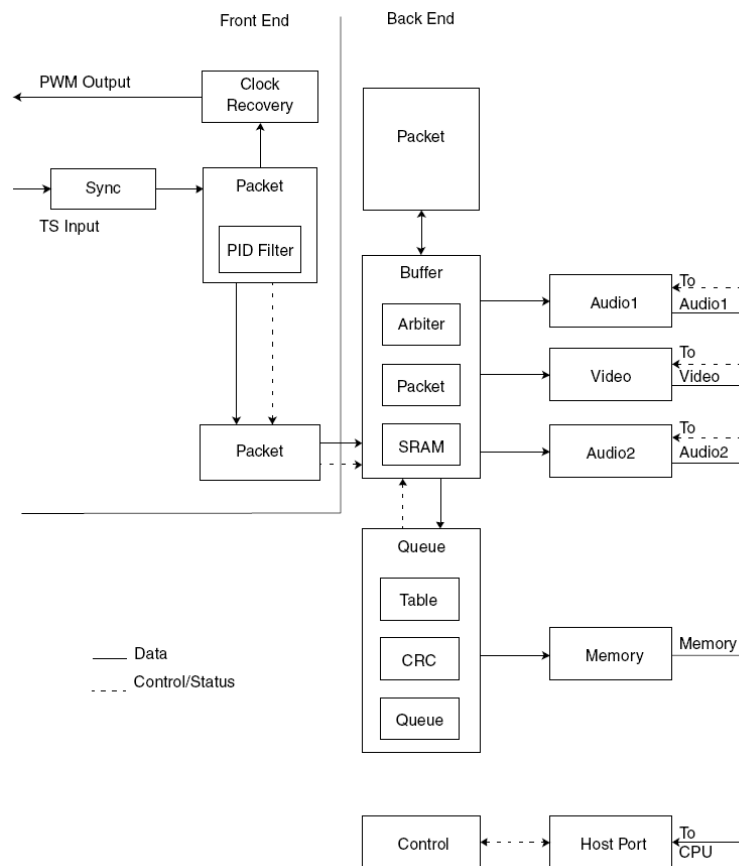


Figura 17: Fluxo de dados do TS através do demultiplexador presente nos chipsets IBM39MPEGCS24PFA16C e IBM39MPEGCS24DPFA16C (1999).

Indicadores de status, que representam as informações analisadas gramaticalmente, são enviados a diante com o pacote de transporte completo para o carregador de transporte para ser armazenado no *buffer* de pacote. O *buffer* de pacote retém até 10 pacotes de transporte enquanto eles são movidos para os decodificadores e filas de memória. Pacotes para todos os três destinos podem ser movidos concorrentemente.

Os três descarregadores extraem os dados do *buffer* de pacote. O descarregador de audio1, o descarregador de audio2 e o descarregador de vídeo enviam os dados para os respectivos decodificadores quando requisitados. O descarregador de memória envia dados para o controlador de memória para a subsequente transferência para a memória

do sistema. Ele pode opcionalmente ser configurado para filtrar seções de tabela e realizar verificação de CRC dos dados de seção.

A porta principal é usada pelo processador principal para configurar, monitorar e controlar a operação do módulo de transporte.

2.3.3 SAA7214 (2001) – Philips

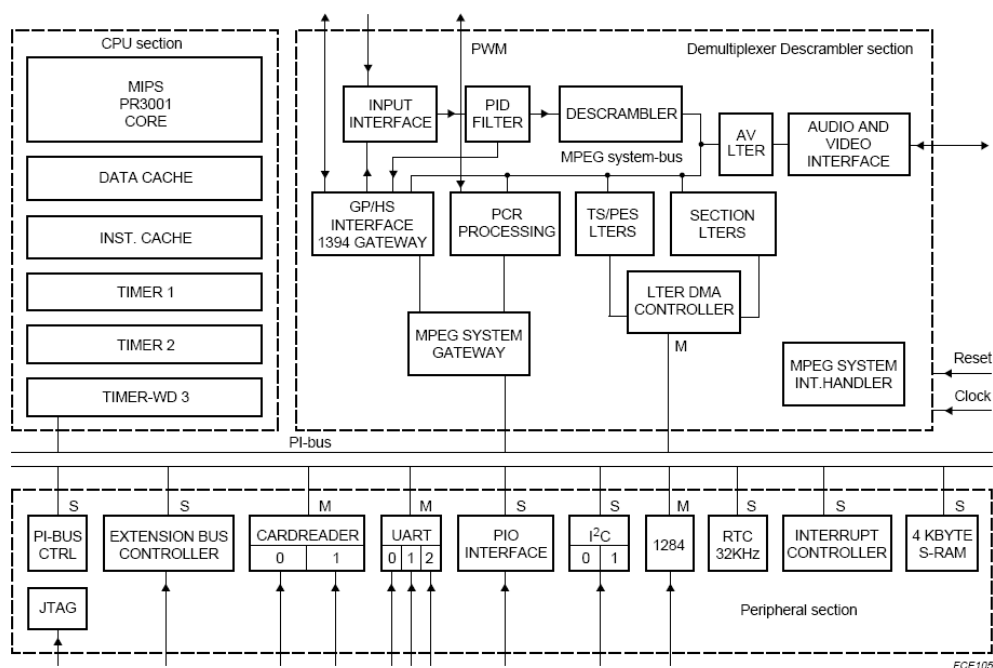


Figura 18: Diagrama de blocos do SAA7214 (2001).

O chip isolado SAA7214 da Philips é um demultiplexador que possui uma arquitetura que inclui três seções, como apresentado na figura 18: a seção da CPU (MIPS PR3001 RISC CPU), a seção demultiplexador/ desembaralhador e a seção de periféricos.

Na seção demultiplexador/desembaralhador existem os módulos de interface de entrada, filtro de PID, desembaralhador, filtros de áudio e vídeo, interface de áudio e vídeo, filtros de seção, filtros de TS/PES, processamento de PCR, controlador de DMA de filtro, *gateway* de GP/HS para interface 1394, *gateway* de sistema MPEG e um manipulador de sistema MPEG. Essa seção é responsável por fazer:

- a análise gramatical de TS, PS e fluxos de dados proprietários;
- o desembaralhamento;

- a filtragem de seção baseada em até 32 PIDs diferentes;
- a filtragem de TS/PES com quatro filtros para recuperação de dados a nível de TS e PES;
- o DMA a uma memória externa baseado no armazenamento de 32 sub-fluxos de seção e quatro sub-fluxos de dados TS/PES;
- o gerenciamento da base de tempo do sistema;
- a temporização de PTS/DTS por dois temporizadores;
- a filtragem GP/HS que serve de entrada alternativa a partir de dispositivos 1394.

2.3.4 SAA7219 (2001) – Philips

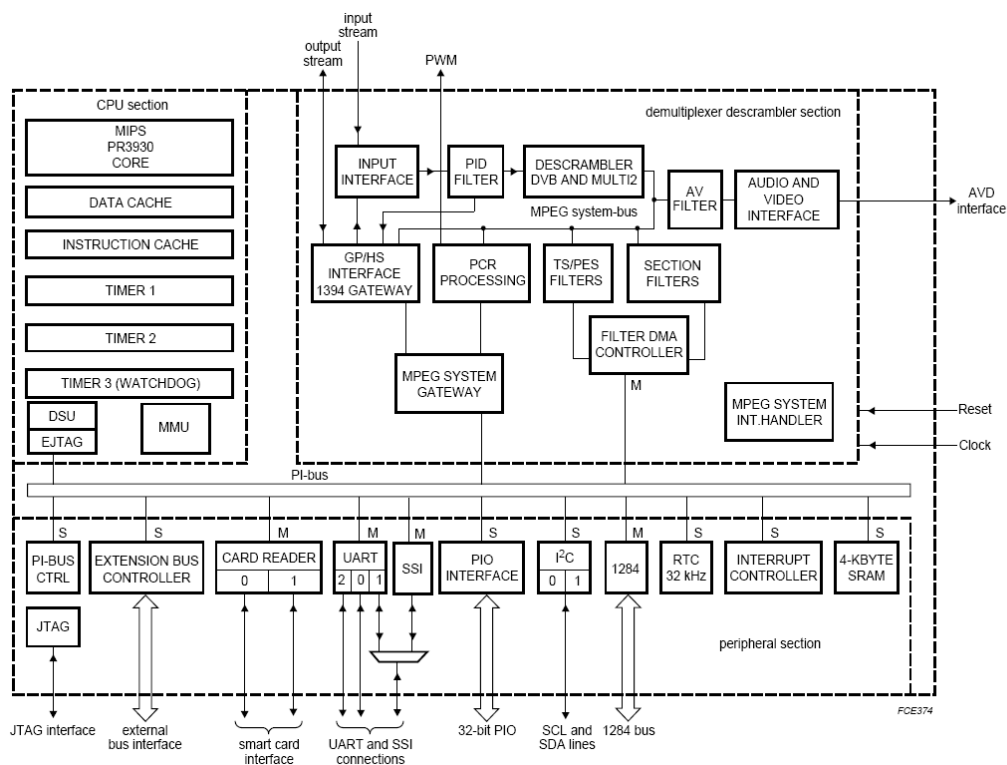


Figura 19: Diagrama de blocos do SAA7219 (2001).

O chip isolado SAA7219 da Philips é um demultiplexador semelhante ao SAA7214. Sua arquitetura é apresentada na figura 19. As diferenças são a CPU que é um MIPS PR3930 RISC, o suporte a desembaralhamento DVB e MULTI2, e algumas interfaces de periféricos.

2.3.5 SAA7240 (2001) – Philips

O chip isolado SAA7240 da Philips, apresentado na figura 20, é um demultiplexador semelhante aos anteriores, com a diferença na arquitetura da seção demultiplexador/desembaralhador que passou a se chamar de processador de sistema MPEG-2 (MSP – *MPEG-2 System Processor*). São cinco blocos funcionais nessa seção: roteador de E/S, PWM, demultiplexador e desembaralhadores, interface de áudio e vídeo, e controlador de *buffer*.

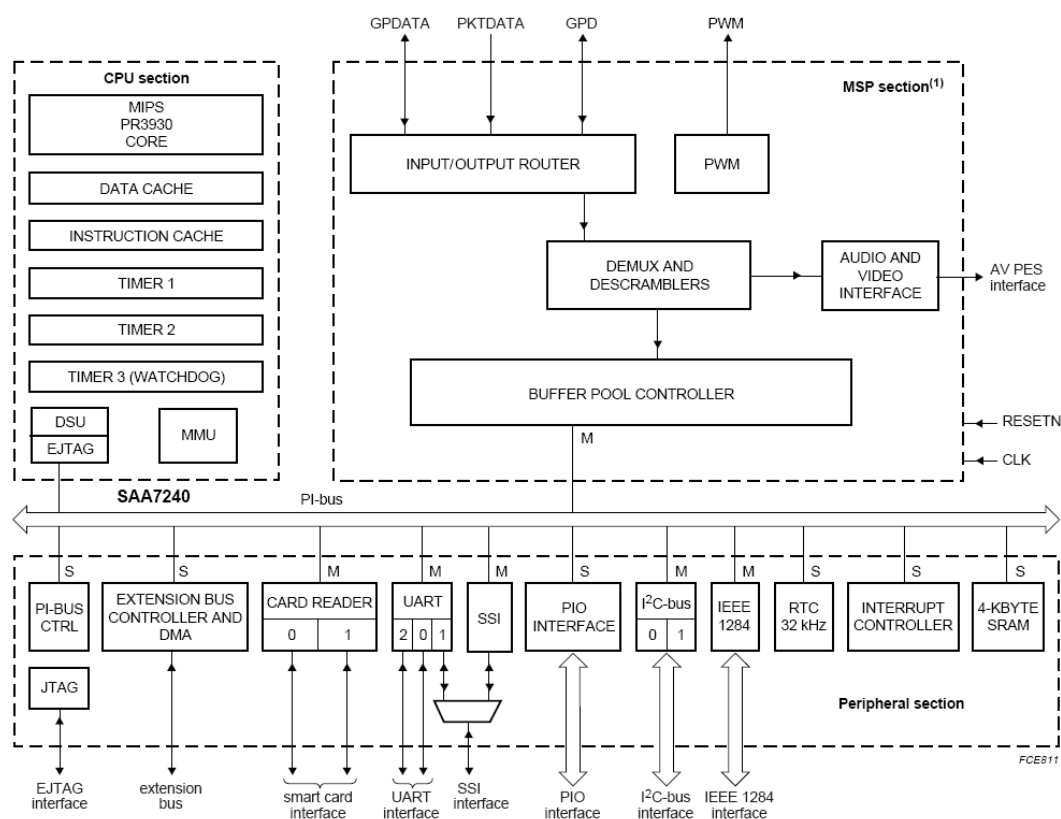


Figura 20: Diagrama de blocos do SAA7240 (2001).

A seção de processador de sistema MPEG-2 é responsável por:

- dar suporte a várias combinações de interfaces de E/S de fluxo de transporte;
- filtragem de cabeçalho de pacotes TS/PES através de 4 filtros;

- filtragem de seção com 32 filtros baseada em um número flexível de condições de filtro (para recuperação de dados PSI, SI, privados, EPG, etc);
- filtragem de ECM/EMM através de 7 filtros;
- DMA a uma memória externa baseado no armazenamento de 32 sub-fluxos de seção e quatro sub-fluxos de dados TS/PES e quatro cabeçalhos de pacote TS/PES;
- gerenciamento da base de tempo do sistema, temporização de PTS/DTS por dois temporizadores;
- filtragem GP/HS que serve de entrada alternativa a partir de dispositivos 1394.

2.3.6 S5H2010 (2003) – Samsung

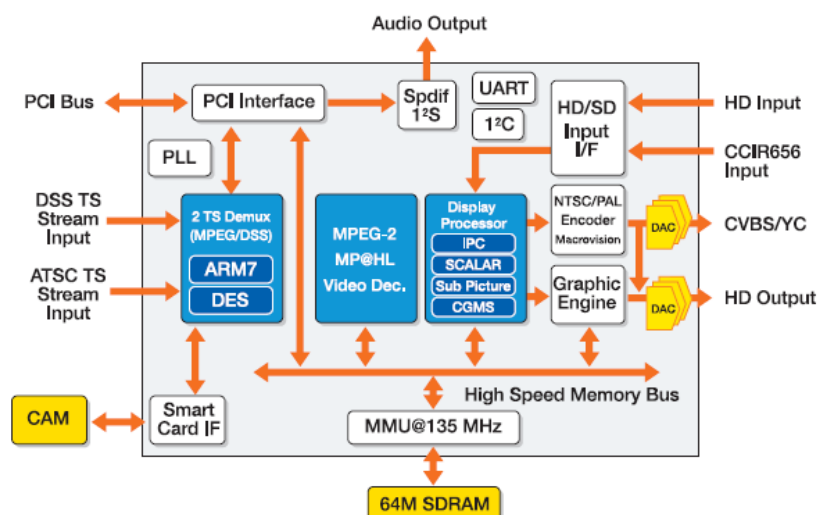


Figura 21: Diagrama de blocos do S5H2010 (2003).

O chipset S5H2010 da Samsung possui uma arquitetura, apresentada na figura 21, que inclui um demultiplexador de transporte (ARM7 e desembaralhador), decodificadores de áudio e vídeo, processador gráfico de apresentação, codificador de vídeo NTSC/PAL, interface de controle e interfaces para periféricos.

O demultiplexador de fluxo de transporte possui uma configuração para demultiplexação em software através de um ARM7, possui um desembaralhador interno, pode receber simultaneamente o máximo de 48 PIDs, faz verificação de CRC, transmite pacotes de vídeo para um SDRAM externa através de DMA, transmite pacotes de áudio

para a memória de uma CPU externa através de uma interface PCI, transmite apenas as informações desejadas a partir de um pacote PSI para uma CPU externa e possui um circuito de recuperação de relógio interno para recuperação de relógio de programa.

O bloco demultiplexador de fluxo de transporte, ilustrado na figura 22, possui o papel de transmitir apenas os dados e informações necessárias para a memória com a filtragem a partir da variedade de informações presentes no fluxo de transporte. O fluxo de transporte entra no módulo por meio da detecção de sincronização. Os pacotes desejados são filtrados pelo PID e enviados para o *buffer* de fluxo de transporte. O processo de demultiplexagem é realizado por software através do envio do pacote por meio da memória para o ARM7 presente. O ARM7 possui modo de de-multiplexagem que depende do tipo de dados (PES, ES, ...). Os dados são então enviados para os decodificadores de áudio e vídeo. A filtragem de seção, com a filtragem de dados das seções de tabela, são transferidas para a CPU externa.

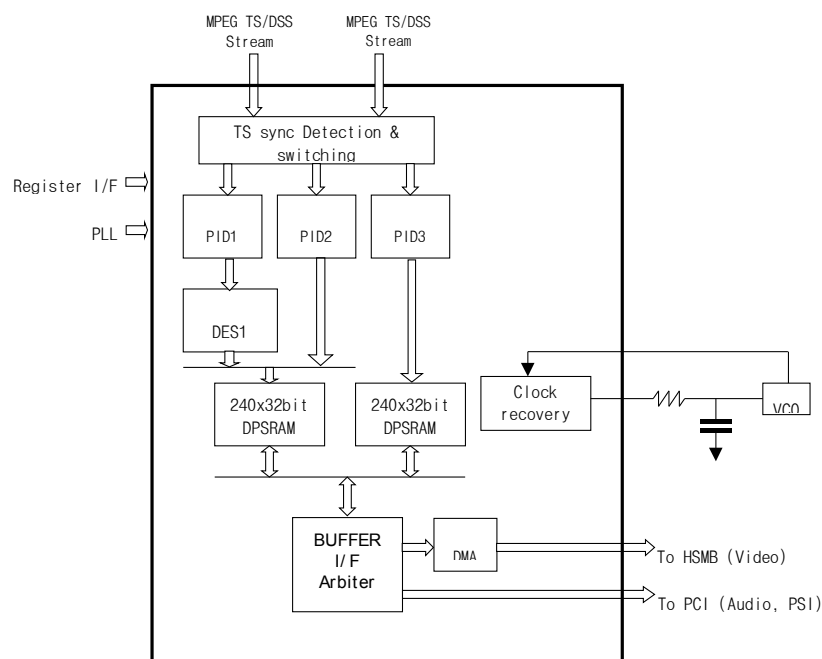


Figura 22: Diagrama de blocos do demultiplexador presente no chipset S5H2010 (2003).

2.3.7 STB7100 (2005) – ST Microelectronics

O single-chip STB7100 da ST Microelectronics utiliza uma arquitetura, apresentada

na figura 23, que inclui uma CPU SH4-202, duas CPUs ST231 para decodificação de áudio e vídeo, demultiplexador de fluxo de transporte, desembaralhador, decodificação de fluxos SD e HD, processador gráfico de apresentação, codificador de vídeo, e interfaces para periféricos.

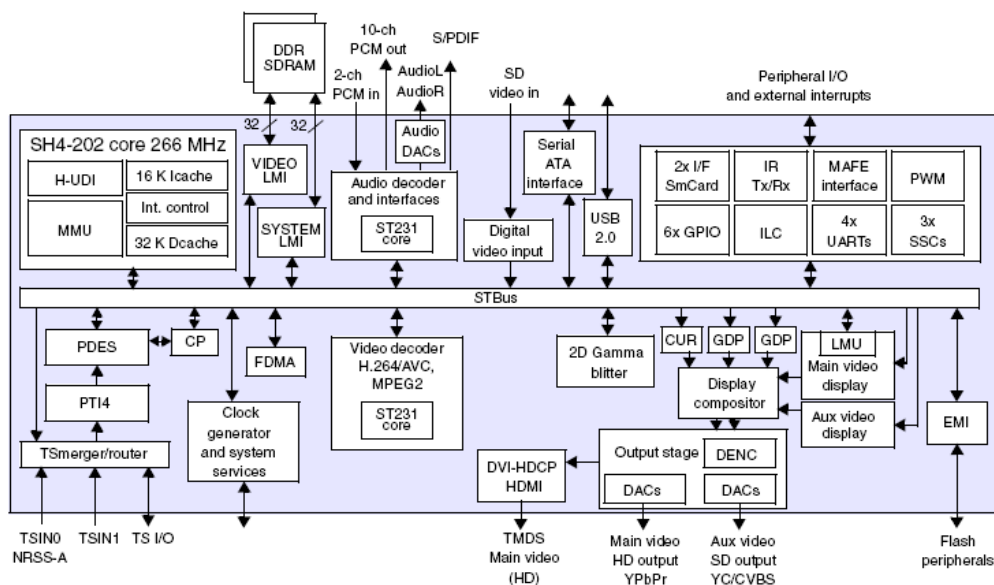


Figura 23: Diagrama de blocos do STB7100 (2005).

Os fluxos de transporte são recebidos e processados pelo subsistema de fluxo de transporte. Os fluxos PES e dados de seção resultantes são armazenados em buffers de memória em uma SDRAM DDR anexa à interface local de memória.

Um controlador DMA flexível realiza a análise do PES e inicia a detecção de código e roteia os fluxos elementares para os buffers de áudio e vídeo da SDRAM DDR. Um decodificador de vídeo MP@HL/H.264 decodifica fluxos de vídeo HD ou SD. A decodificação de vídeo e a combinação PCM é realizado por um decodificador de áudio que tem sua saída através de interfaces S/PDIF e PCM. A saída de áudio pode ser também através do codificador de vídeo.

O subsistema de fluxo de transporte, apresentado na figura 24, inclui o unificador/roteador de fluxo de transporte e uma interface programável de transporte (PTI - *Programmable Transport Interface*).

Os fluxos de transporte entram por uma das três interfaces. Duas dessas entradas são paralelas, que também podem ser configuradas para entrada serial se necessário. A

terceira é uma interface paralela bidirecional que pode ser configurada como entrada ou saída.

As cinco entradas e duas saídas do unificador/roteador de fluxo de transporte permite que qualquer das três entradas externas serem roteadas para o PTI. Uma quarta entrada é fornecida para o roteamento interno de fluxos de transporte armazenados na memória para o PTI. Uma quinta entrada recebe um fluxo de transporte parcial ou total criado pelo PTI que pode sair do STB7100 por meio da interface bidirecional quando configurado como saída. Isso permite que fluxos de transporte serem enviados para um D-VCR por meio de um controlador da camada de enlace IEEE 1394. O roteamento para as duas saídas pode ser concorrente.

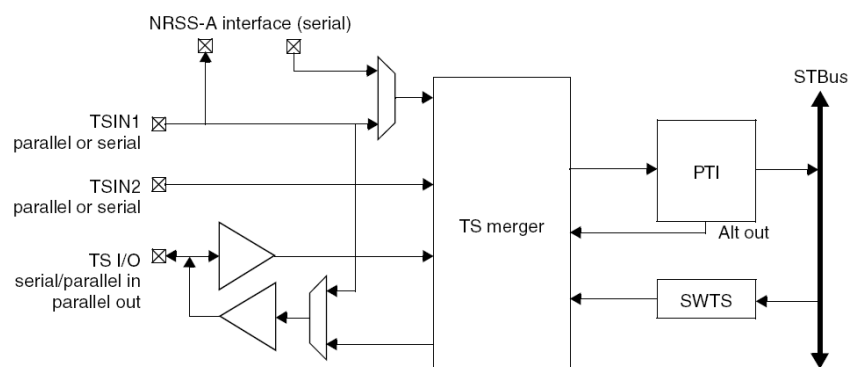


Figura 24: Subsistema de fluxo de transporte do STB7100 (2005).

Uma interface NRSS-A é disponibilizada para o roteamento de fluxos de transporte serial a partir e para um módulo de acesso condicional compatível.

O unificador/roteador de fluxo é também capaz de unir três fluxos de entrada em um fluxo de transporte e remetê-lo para o PTI permitindo isso para processar três fontes independentes de fluxo de transporte ao mesmo tempo.

O PTI realiza a filtragem de PID, demultiplexação, desembaralhamento, e filtragem de dados em até três fluxos de transporte ao mesmo tempo. O PTI extrai PCRs com marcas de tempo e as torna disponível para a CPU para a recuperação de relógio e sincronização de áudio e vídeo.

Os dados PES são transferidos pela DMA para buffers de memória. Os dados de seção são transferidos pela DMA para buffers separados para o posterior processamento pela CPU. O PTI pode também extrair informação indexada e, então, transferir pacotes,

usando o DMA, para um buffer intermediário para armazenamento.

O PTI realiza filtragem de PID para selecionar o áudio, vídeo e pacotes de dados a serem processados. Até 96 posições de PID são suportadas.

O PTI possui um módulo de filtragem de seção de 48 x 16 bytes. Três modos de filtragem são disponibilizados: modo *wide match* com filtros de 48 x 16 bytes, modo *long match* com filtros de 96 x 8 ou modo positivo/negativo com filtros 48 x 8 bytes com filtragem positiva/negativa a nível de bit.

A seções associadas são transferidas para buffers de memória para processamento por software.

Quando o PTI é necessário para a saída de um fluxo de transporte, ele pode colocar na saída o fluxo de transporte completo, ou pacotes selecionados filtrados pelo PID. Um contador de latência é fornecido para garantir que a temporização do pacote é preservada. Os pacotes podem também ser substituídos.

2.3.8 STI7101 (2008) – ST Microelectronics

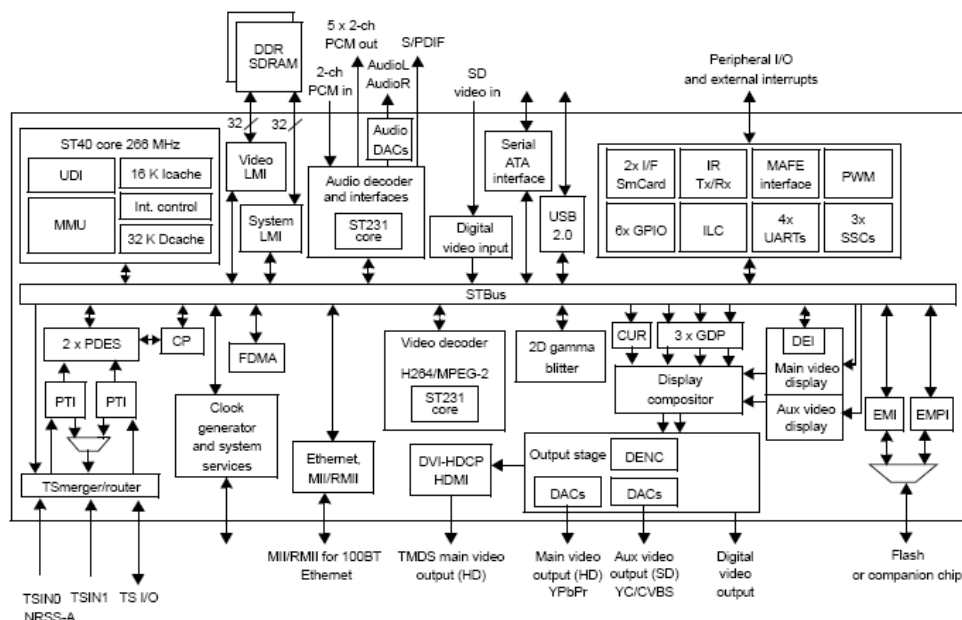


Figura 25: Diagrama de blocos do STI7101 (2008).

O single-chip STI7101 da ST Microelectronics possui uma arquitetura , apresentada na figura 25, que possui uma CPU ST40, duas CPUs ST231 para decodificação de áudio e vídeo, módulo de filtragem de transporte e desembaralhamento, processador gráfico de

apresentação, codificador de vídeo, e interfaces para periféricos.

O subsistema de transporte possui um unificador/roteador de fluxos de transporte que possui duas entradas seriais/paralelas, uma interface bidirecional e pode combinar três fluxos de transporte, e duas interfaces de transporte programáveis (PTIs) que possuem um demultiplexador de fluxo de transporte e desembaralhadores para cada um.

2.3.9 Resumo dos Chips Comerciais apresentados

Dentre os chips comerciais estudados que envolvem a demultiplexação de TS, os classificados como soluções integradas em um único chip (*single chip*), que englobam funcionalidades além da demultiplexação do fluxo MPEG-2 TS foram os da empresa ST Microelectronics; como soluções baseadas em múltiplos chips (*chipset*), com funcionalidades além da demultiplexação, com interfaces para controle por processador externo, foram os das empresas IBM e Samsung; e como soluções baseadas em chips isolados, que desempenham a tarefa específica de de-multiplexagem, foram os da empresa Philips.

Na tabela 2, é apresentado o quadro comparativo dos chips comerciais estudados nesta dissertação.

Tabela 2: Quadro comparativo dos chips comerciais apresentados nesta dissertação.

Ano	Dispositivos	Empresa	Tipo
1997	SAA7205H	Philips	chip isolado
1999	IBM39MPEGCS24PFA16C e IBM39MPEGCS24DPFA16C	IBM	chipset
2001	SAA7214	Philips	chip isolado
2001	SAA7219	Philips	chip isolado
2001	SAA7240	Philips	chip isolado
2003	S5H2010	Samsung	chipset
2005	STB7100	ST Microelectronics	single-chip
2008	STI7101	ST Microelectronics	single-chip

3 Especificação, Modelagem e Validação em Alto Nível do Demultiplexador Proposto

Neste capítulo serão abordadas questões relativas à especificação e modelagem do demultiplexador proposto em alto-nível, em linguagem C, seguindo-se da correspondente validação. Seguindo a metodologia de desenvolvimento **Top-Down**, o propósito deste capítulo é apresentar as etapas que precederam a implementação no nível RTL. Está dividido em três partes: detalhamento da especificação, na forma de três vistas (funcional, arquitetural e de verificação); a parte de implementação do modelo estruturado do nível da interface aos blocos elementares de processamento de bit; e a parte da validação, submetendo os arquivos de saída do modelo a programas reprodutores de áudio e vídeo digitais.

3.1 Especificação do Demultiplexador Proposto

As **funções** do demultiplexador são: colocar os fluxos de vídeo na saída de vídeo, os de áudio na saída de áudio, os dados PES na saída de dados, os dados PSI na saída de informação do sistema, os dados de sincronização de áudio e vídeo na saída de sincronização e os dados PCR na saída de recuperação de relógio.

A especificação apresentada a seguir é dividida em três vistas, de acordo com as orientações de projeto tomadas durante o desenvolvimento do demultiplexador. São elas: a especificação funcional, a especificação arquitetural e a especificação de verificação.

Na **especificação funcional**, são apresentadas suas diferentes funções e o levantamento dos correspondentes requisitos necessários.

Na **especificação arquitetural**, estão descritos o nível da interface, o conjunto de blocos funcionais em que foi particionado o problema e os sub blocos de cada um.

Na **especificação de verificação** deste modelo de alto nível, o foco foi na definição de um ambiente que garantisse a verificação do funcionamento correto (ou não) do demultiplexador. No escopo deste trabalho, foi considerado como funcionamento correto a reprodução sem erro acusado pelos programas reprodutores de áudio e vídeo digitais adotados.

3.1.1 Especificação funcional

As características deste módulo demultiplexador são:

- Estar de acordo com a norma ISO/IEC 13818-1 (2000) que contém a especificação do fluxo de transporte;
- Total conformidade com as normas ABNT NBR 15602-3 (2007), ABNT NBR 15603-1(2007), ABNT NBR 15603-2(2007) e ABNT NBR 15603-3(2007);
- Configuração de filtragem para PIDs de áudio, vídeo, dados , PCR e de seção;
- Detecção de PID e descarte dos pacotes do fluxo de transporte que não foram detectados logo na chegada;
- Filtragem de seção PSI/SI;
- Detecção de PCR;
- Extração de áudio, vídeo e dados PES;
- Extração de PTS e DTS;
- Sincronização de cabeçalho TS;
- Detecção de CRC-32 para pacotes PSI/SI;
- Memória interna do tipo fila para entrada e saída de dados.

A função do módulo demultiplexador é analisar a gramática dos cabeçalhos dos pacotes TS e encaminhar as informações do pacote para o espaço de memória ou E/S designada pela análise.

Os sinais que entram no módulo demultiplexador são **classificados** de acordo com o **tipo de pacote TS**. A figura 26 ilustra os três tipos de pacotes TS existentes.

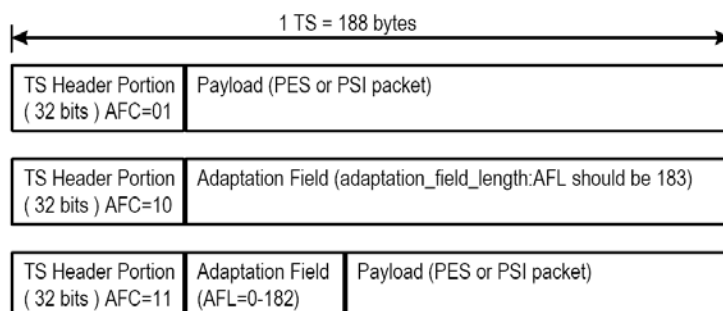


Figura 26: Tipos de pacote TS.

Os dados do fluxo de transporte são empacotados com uma largura fixa de 188 bytes. O **pacote TS** contém 32 bits de informação de **cabeçalho** com um **campo de adaptação opcional**. O campo de adaptação é especificado pelos bits de controle do campo de adaptação presente no cabeçalho do TS. Os dados de sincronização PCR podem ser extraídos do campo de adaptação, para então serem encaminhados para a saída correta. A parte relativa à **payload** contém dados de áudio, vídeo, de sincronização e de informação específica de programa, que pode ser tanto um pacote PES como um pacote PSI/SI, como definido no padrão ISO/IEC 13818-1 (2000).

O **pacote PES** possui uma parte relacionada ao **cabeçalho**, um **cabeçalho opcional** onde podem estar presentes os dados PTS e DTS, e uma parte relacionada ao **fluxo elementar**. Os dados do cabeçalho PES são responsáveis pela identificação do tipo e número do fluxo elementar. O encaminhamento do fluxo elementar para a saída correta é feita após essa identificação. Áudio, vídeo e dados estão na forma de fluxo elementar.

A **seção PSI/SI**, ou pacote PSI/SI, possui dois formatos: o **formato geral** e o **formato estendido**. O formato geral possui uma parte relacionada ao cabeçalho e uma para os dados. O formato estendido possui cabeçalho, dados e CRC. Os dados do cabeçalho PSI/SI identifica o tipo de dados PSI/SI que está presente na parte de dados do pacote. Após essa identificação, os dados PSI/SI são encaminhados para a saída adequada.

A recepção de pacotes TS é a primeira tarefa realizada pelo módulo demultiplexador. Os seguintes passos são implementados pelo **mecanismo de recepção** do pacote do fluxo de transporte:

- Detecção do cabeçalho e sincronização do pacote. Um byte do fluxo com valor igual 0x47 é detectado, funcionando com um cabeçalho do pacote. O módulo demultiplexador conta mais 187 bytes e verifica o próximo byte também tem valor igual a 0x47. Em caso positivo, o pacote é considerado válido e tem seu cabeçalho sincronizado.
- Comparação do PID e armazenamento no buffer de entrada do pacote. Os PIDs do fluxo de entrada são comparados com os PIDs do programa selecionado pelo sistema receptor. Os pacotes são transferidos para o buffer de entrada se considerados válidos e esperam pela sua decodificação, ou são descartados, se

forem considerados inválidos.

A Tabela 3 apresenta os sinais necessários à interface do demultiplexador:

Tabela 3: Interfaces do demultiplexador.

Nome	Direção	Polaridade	Registrador	Largura (bits)	Descrição
Clock	Entrada	Ativo no Alto	Não	1	Relógio para sincronização
Reset	Entrada	Ativo no Baixo	Não	1	Reset para reiniciar operação
TS	Entrada	Ativo no Alto	Sim	8	Entrada do TS multiplexado
PID de Programa	Entrada	Ativo no Alto	Sim	13	Programa selecionado pelo usuário (PID do programa)
PCR	Saída	Ativo no Alto	Sim	8	PCR extraído do TS para recuperação de relógio
Áudio	Saída	Ativo no Alto	Sim	8	Áudio extraído do TS para o decodificador de áudio
Vídeo	Saída	Ativo no Alto	Sim	8	Vídeo extraído do TS para o decodificador de vídeo
Dados PES	Saída	Ativo no Alto	Sim	8	Dados extraídos para processamento
PTS/DTS	Saída	Ativo no Alto	Sim	8	PTS/DTS extraído do TS para sincronização de áudio e vídeo
PSI/SI	Saída	Ativo no Alto	Sim	8	Informações do sistema extraído para tabelas SI

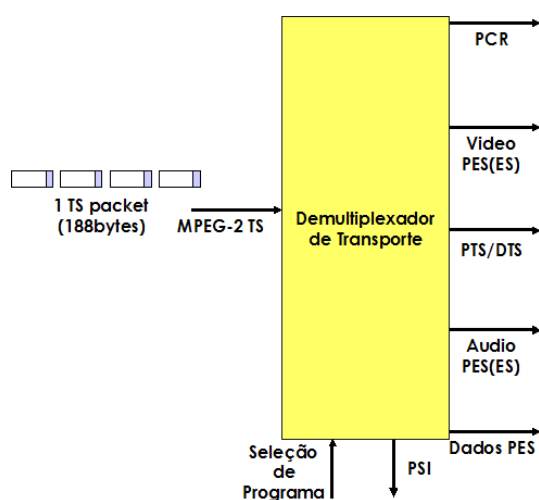


Figura 27: Demultiplexador de referência.

Considerando-se que o alvo final do projeto desenvolvido ser a implementação de um módulo em forma de propriedade intelectual (IP) e sua possível utilização em SoCs, optou-se, já neste alto nível por ampliar o modelo de referência para contemplar uma interface compatível com um sistema de interconexão de uso corrente nessas aplicações.

Para atender este **requisito de comunicação**, próprio de projetos que utilizam o método SoC, foi selecionado o protocolo AMBA AXI (ARM IHI 0022B, 2004), extensamente utilizado em implementações de sistemas embarcados utilizando o micro processador comercial ARM, evitando criar um novo protocolo de comunicação. Esse protocolo apresenta flexibilidade na forma como trata a comunicação entre dispositivos que, ora querem enviar informações, ora precisam de informações e precisam requisitá-las.

Outro fato que influenciou na escolha do protocolo AMBA AXI, é a sua utilização no projeto BSMILC, que vem sendo desenvolvido no LASIC/LASID (Laboratório de Arquitetura, Sistemas Integráveis e Circuitos e Laboratório de Sistemas Digitais) da Universidade Federal da Paraíba – UFPB – dentro do consórcio *Brazil-IP*. A adesão a esse padrão já em uso por outro projeto do LASIC, possibilitou um domínio mais ágil do protocolo enquanto apontou para uma direção conjunta para ambos projetos. A presença constante de IPs ARM em dispositivos móveis embarcados e de TV Digital pesou também na decisão de adoção desse protocolo.

3.1.2 Especificação arquitetural

O modelo arquitetural foi construído a partir da separação do modelo funcional em partes capazes de processar **funcionalidades** relativas a:

- Sincronização do cabeçalho dos pacotes TS;
 - Detecção e armazenamento temporário de pacotes com PIDs selecionados pelo usuário do demultiplexador;
 - Armazenagem temporária dos cabeçalhos dos pacotes de TS, PES e PSI/SI durante processamento. O demultiplexador não visa armazenar informações da
-

payload desses pacotes, e sim repassar esses dados adequadamente;

- Filtragem de seção de pacotes PSI/SI;
- Detecção de CRC32 para pacotes PSI/SI;
- Necessidade de desempenho para grandes quantidades de pacotes, com PIDs selecionáveis;

O particionamento e agrupamento de funcionalidades permitiu a definição dos componentes que estão presentes na arquitetura apresentada no capítulo 4, mantendo como objetivos separar funções independentes em blocos, maximizar a coesão intra blocos, minimizar as conexões inter blocos.

O **modelo proposto** para o demultiplexador incluiu os seguintes componentes, organizados em 2 níveis hierárquicos internos:

- Buffer de entrada;
 - Filtro de Pacotes TS;
 - Filtro de Pacote de Programa;
 - Fila de Entrada de Pacote TS;
 - Analisador de Pacotes TS;
 - Variável armazenadora de pacote TS;
 - Analisador de cabeçalho de pacotes TS;
 - Analisador de Campo de Adaptação;
 - Filtro PCR;
 - Fila de saída PCR;
 - Extrator de PES;
 - Analisador de cabeçalho PES;
 - Filtro PTS/DTS;
 - Fila de saída de PTS/DTS;
 - Fila de saída de vídeo;
 - Fila de saída de áudio;
 - Fila de saída de dados PES;
 - Extrator de PSI/SI;
 - Analisador de cabeçalho de PSI/SI;
 - Filtro de seção;
-

- Detector de CRC32;
- Fila interna de CRC32;
- Fila de saída de PSI/SI;
- Variáveis para transferência de dados entre os componentes internos e E/S.

3.1.3 Especificação de verificação

A metodologia de verificação usada foi a **Bottom-Up**, consistindo em três fases: a verificação de **sub-blocos** individuais, dos **blocos** incorporando sub-blocos validados e a verificação do **sistema completo** com a interconexão dos blocos validados.

A ferramenta QuestaSim 6.4, utilizada nesta fase, inclui simulação com suporte a linguagens de verificação de hardware e análise cobertura de código. A inspeção, assim como a análise de taxa de cobertura, também foram usadas como forma de verificação.

Foram verificadas as seguintes **características**:

- Atendimento aos padrões brasileiros de TV digital pelas funções de análise gramatical dos cabeçalhos do TS;
- Atendimento ao padrão AMBA AXI pelas interfaces.

As verificações foram locais para cada função de análise gramatical do TS, com a criação de vetores de teste de tamanho definido, contendo fluxos de transporte para cada tipo de cabeçalho previsto nos padrões e também não previstos.

O **mecanismo de parada**, ou critério de conclusão, por se tratar de testes determinísticos foi a conclusão do processamento do último fluxo de transporte.

3.2 Implementação do modelo funcional hierárquico de alto-nível

Esse modelo de alto-nível é uma “**especificação executável**” e trata de uma abordagem algorítmica, puramente comportamental, sem informações de temporização, contemplando o nível de interface e 2 níveis de hierarquia interna.

Para sua implementação foi utilizada a linguagem de programação C, com os tipos

próprios da linguagem e também a biblioteca *ac_int.h*, desenvolvida pela Mentor Graphics, que faz parte do pacote *Algorithmic C Datatypes* v1.2 (DATATYPES, 2008). Com essa biblioteca, foi possível definir inteiros com tamanhos arbitrários e com precisão ao nível do bit, o que facilitou muito a modelagem de funções digitais.

A criação do modelo comportamental em linguagem C se baseou na especificação do fluxo de transporte – TS presente no padrão ISO/IEC 13818-1 (2000), cuja sintaxe descreve o significado e o formato dos bytes e pacotes de bytes contidos em suas sequências binárias.

O **modelo**, através do bloco *demux*, recebe o TS por meio de um arquivo e inicia a leitura de pacotes TS, chamando o bloco *mpeg_transport_stream*. Esse bloco fica responsável por determinar o **início do pacote** através da análise dos bytes que chegam, através da função *next_bits*, para encontrar o byte de sincronização 0x47.

Uma vez sincronizado, é feita a **análise gramatical do cabeçalho** do pacote TS, onde se determina o PID do pacote, a existência do campo de adaptação e o tipo da *payload*. Essa análise é feita pelo bloco *transport_packet*.

Se for verificada a existência da **camada de adaptação**, o bloco *adaptation_field* fará a análise do seu cabeçalho. E, definido o **tipo da payload**, o *transport_packet* chamará a função de análise da *payload* adequada. A *payload* pode ser um pacote PES ou uma seção PSI/SI.

Caso seja um **pacote PES**, o bloco *PES_packet* será chamado. O campo *stream_id* do cabeçalho vai determinar o **significado do fluxo elementar** presente nesse pacote PES. Ainda no bloco PES, pode haver a possibilidade de um PS estar empacotado. Nesse caso, o bloco *pack_header* é chamado para fazer a análise do cabeçalho do PS presente, utilizando-se também do bloco *system_header* para essa tarefa, caso seja necessário.

Agora, caso a *payload* do pacote TS seja uma **seção PSI/SI**, o bloco *PSI* é que será chamado. As seções contêm informações sobre o sistema e são necessárias para a montagem das tabelas do sistema de informação, como as tabelas PAT, PMT, CAT, NIT, etc. Esse bloco analisa o PID e a informação *table_id* da seção para determinar o **tipo da seção** a ser analisada. De acordo com essas informações, um bloco de seção adequado é chamado. Existem 15 **tipos de seções** diferentes, definidas pela norma brasileira ABNT NBR 15603-2 (2007). Essa norma contém a estrutura de dados e definições da

informação básica da SI. Caso a informação estendida da norma ABNT NBR 15603-3 (2007) seja levada em consideração, o número de seções sobe para 18.

Os blocos das seções ficam responsáveis por chamar os blocos descritores, completando as informações da SI.

Na figura 28, é apresentado o diagrama do bloco demultiplexador com os sinais que são conectados diretamente a ele. O arquivo TS de referência é recebido pelo demultiplexador, são separados todos os fluxos e, dentro do escopo deste trabalho, foram capturados os fluxos de áudio e vídeo.

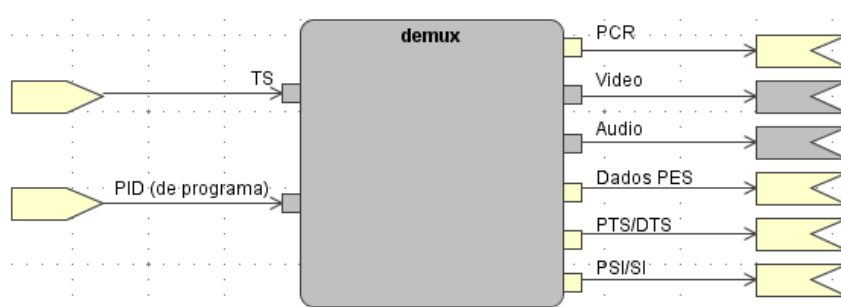


Figura 28: Diagrama do bloco demultiplexador destacando os sinais de entrada e saída.

O bloco demultiplexador é composto pelo nível de interface (`mpeg_transport_stream`) e dois níveis internos de hierarquia: TS (`transport_packet`) e PES (`PES_packet (pack_header (system_header))`) / PSI_SI como é apresentado na figura 29.

Estes níveis hierárquicos correspondem ao esquema de empacotamento utilizado no protocolo MPEG-2 Sistema. Em primeiro lugar, o fluxo elementar proveniente do codificador é empacotado em um Packetized Elementary Stream - PES. Em seguida, o PES é empacotado em um TS ou PS, dependendo da aplicação. O fluxo elementar resultante é então montado pelo encadeamento de cada conteúdo e pela preservação de todas as informações sobre cada tipo, ponto de início e duração. O demultiplexador deve reconhecer todos os limites dos pacotes para processar os campos de dados e extrair as *payloads*. A figura 29 mostra o modelo hierárquico da especificação executável desenvolvido para isso.

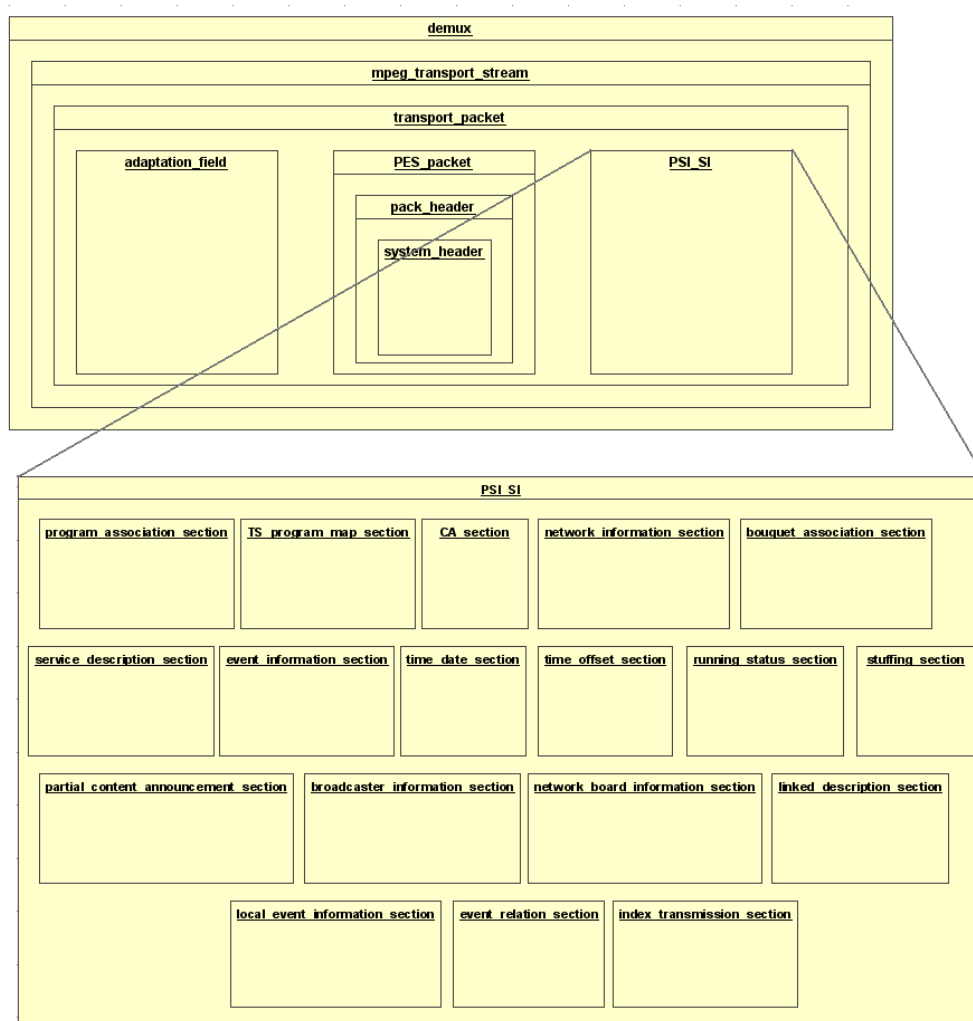


Figura 29: Modelo hierárquico da especificação executável do demux.

Os **descritores** são estruturas que podem ser usadas para estender as definições de programas e elementos de programa. Eles aparecem dentro das seções. Os 57 descritores que devem estar presentes no sistema de informação, definidos pela norma ABNT NBR 15603-1, são listados a seguir. Os descritores com asterisco não possuem suas estruturas descritas nas normas ABNT NBR 15603-2 (2007) e ABNT NBR 15603-3 (2007), mas sim, em suas referências como as normas ARIB STD-B10 (2008) e ETSI EN 300 468 (2003).

- *conditional_access_descriptor* *
- *copyright_descriptor* *
- *network_name_descriptor*
- *service_list_descriptor*
- *stuffing_descriptor*
- *terrestrial_delivery_system_descriptor*

- *bouquet_name_descriptor*
- *service_descriptor*
- *country_availability_descriptor* *
- *linkage_descriptor*
- *NVOD_reference_descriptor* *
- *time_shifted_service_descriptor* *
- *short_event_descriptor* *
- *extended_event_descriptor* *
- *time_shifted_event_descriptor* *
- *component_descriptor* *
- *mosaic_descriptor* *
- *stream_identifier_descriptor* *
- *CA_identifier_descriptor* *
- *content_descriptor* *
- *parental_rating_descriptor*
- *hierarchical_transmission_descriptor*
- *digital_copy_control_descriptor*
- *emergency_information_descriptor*
- *data_component_descriptor*
- *system_management_descriptor*
- *local_time_offset_descriptor* *
- *audio_component_descriptor*
- *target_region_descriptor* *
- *hyperlink_descriptor*
- *data_content_descriptor*
- *video_decode_control_descriptor*
- *basic_local_event_descriptor*
- *reference_descriptor*
- *node_relation_descriptor*
- *short_node_information_descriptor*
- *STC_reference_descriptor*
- *partial_reception_descriptor*
- *series_descriptor*
- *event_group_descriptor*
- *SI_parameter_descriptor*
- *broadcast_name_descriptor*
- *component_group_descriptor*
- *SI_prime_TS_descriptor*
- *board_information_descriptor*
- *LDT_linkage_descriptor*
- *connected_transmission_descriptor*
- *TS_information_descriptor*
- *extended_broadcaster_descriptor*
- *logo_transmission_descriptor*
- *content_availability_descriptor*
- *carousel_compatible_composite_descriptor*
- *conditional_playback_descriptor*
- *AVC_video_descriptor*
- *AVC_timing_and_HRD_descriptor*
- *service_group_descriptor* *

3.3 Implementação dos testbenches de verificação para o modelo de alto-nível

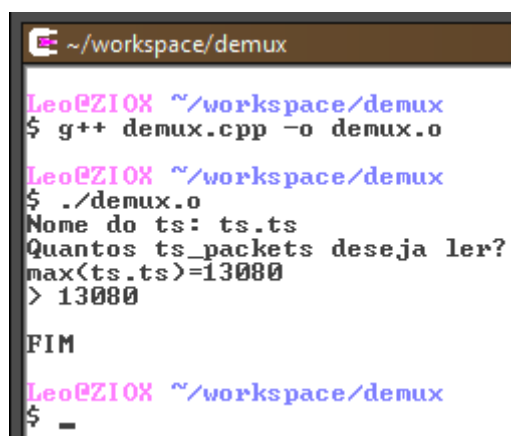
Um **testbench** de verificação a partir do modelo de alto-nível foi desenvolvido para validar e refinar o próprio modelo face às especificações contidas nas normas e as

subsequentes implementações do objeto em questão, a saber: nível RTL, síntese lógica e mapeamento(s) tecnológico(s). Esse ambiente fornece um mecanismo automático para verificar (e apontar as possíveis falhas) se as repostas aos estímulos funcionais estão ou não corretas estaticamente (RTL e síntese lógica) e também dinamicamente, nos processos de *backannotation* que ocorrem nas fases de mapeamento tecnológico.

Este modelo de alto nível aborda o nível hierárquico do *transport_packet*, onde os três tipos de pacotes descritos na figura 26 da página 73 devem estar contemplados no correspondente testbench assim como o resultado esperado para cada tipo. A partir da detecção da ocorrência de um pacote completo (188 bytes) a análise do cabeçalho, conduzirá ao correto tratamento do pacote detectado segundo o tipo deste.

Neste trabalho foi montado um **arquivo de referência** contendo os três tipos de pacotes descritos na figura 26. Este arquivo foi gerado com a ferramenta TSReader v2.6.41(TSREADER, 2009).

A **validação** neste nível foi feita através da submissão do TS produzido à especificação executável – figura 30 – e sua correta reprodução por decodificadores de áudio e vídeo digitais.



```
Leo@ZIOX ~/workspace/demux
$ g++ demux.cpp -o demux.o
Leo@ZIOX ~/workspace/demux
$ ./demux.o
Nome do ts: ts.ts
Quantos ts_packets deseja ler?
max(ts.ts)=13080
> 13080
FIM
Leo@ZIOX ~/workspace/demux
$ -
```

Figura 30: Aplicação do fluxo de transporte da validação na especificação de alto nível do demultiplexador proposto.

Através da ferramenta Media Player Classic v6.4.9.1, é possível assistir o arquivo TS original com o vídeo e o áudio. Através desta mesma ferramenta, também é possível assistir o arquivo de vídeo demultiplexado sem o áudio, atuando como decodificador de vídeo. A figura 31 ilustra a execução do arquivo de referência pela ferramenta citada. E,

através da ferramenta VLC media player v0.9.4 (VIDEOLAN, 2008), é possível ouvir o arquivo de áudio demultiplexado, atuando como decodificador de áudio. Uma ferramenta que também foi bem útil nesse processo foi a MPEG TS Utils v1.1.4 – DEMO, apresentada na figura 32, que é capaz de apresentar a sintaxe de um TS, e ajudou muito na compreensão do problema tratado.



Figura 31: Fluxo de transporte usado na validação.

Offset	syn...	trans...	payload_unit...	transport_priority	PID	transport_scrambling...	adaptation...	continuity...	PSI	Info
\$0000000000000000	\$47	0	1	0	\$0000	0	1	0	PAT	PLD
\$0000000000000000	\$47	0	1	0	\$0020	0	1	0	PMT	PLD
\$0000000000000178	\$47	0	1	0	\$0011	0	1	0	SDT-ATS	PLD
\$0000000000000234	\$47	0	0	0	\$022C	0	1	7		PLD
\$00000000000002F0	\$47	0	0	0	\$022C	0	1	8		PLD
\$00000000000003AC	\$47	0	1	0	\$0238	0	3	0		AF+PLD
\$0000000000000468	\$47	0	0	0	\$0238	0	1	1		PLD
\$0000000000000524	\$47	0	0	0	\$0238	0	1	2		PLD
\$00000000000005E0	\$47	0	0	0	\$0238	0	1	3		PLD
\$000000000000069C	\$47	0	0	0	\$0238	0	3	4		AF+PLD
\$0000000000000758	\$47	0	1	0	\$0238	0	3	5		AF+PLD
\$0000000000000814	\$47	0	0	0	\$0238	0	1	6		PLD
\$00000000000008D0	\$47	0	0	0	\$0238	0	1	7		PLD
\$000000000000098C	\$47	0	0	0	\$0238	0	1	8		PLD
\$0000000000000A48	\$47	0	0	0	\$0238	0	3	9		AF+PLD
\$0000000000000B04	\$47	0	1	0	\$0238	0	3	10		AF+PLD
\$0000000000000BC0	\$47	0	0	0	\$0238	0	1	11		PLD
\$0000000000000C7C	\$47	0	0	0	\$0238	0	1	12		PLD
\$0000000000000D38	\$47	0	0	0	\$0238	0	1	13		PLD
\$0000000000000DF4	\$47	0	0	0	\$0238	0	3	14		AF+PLD
\$0000000000000EB0	\$47	0	1	0	\$022C	0	1	9		PLD
\$0000000000000F8C	\$47	0	0	0	\$022C	0	1	10		PLD
\$0000000000001028	\$47	0	0	0	\$022C	0	1	11		PLD
\$00000000000010E4	\$47	0	0	0	\$022C	0	1	12		PLD
\$00000000000011A0	\$47	0	1	0	\$0238	0	3	15		AF+PLD
\$000000000000125C	\$47	0	1	0	\$020A	0	2	0		AF
\$0000000000001318	\$47	0	0	0	\$0238	0	1	0		PLD
\$00000000000013D4	\$47	0	0	0	\$0238	0	1	1		PLD
\$0000000000001490	\$47	0	0	0	\$0238	0	1	2		PLD
\$000000000000154C	\$47	0	0	0	\$0238	0	3	3		AF+PLD
\$0000000000001608	\$47	0	1	0	\$0238	0	3	4		AF+PLD
\$00000000000016C4	\$47	0	0	0	\$0238	0	1	5		PLD
\$0000000000001780	\$47	0	0	0	\$0238	0	1	6		PLD
\$000000000000183C	\$47	0	1	0	\$0000	0	1	1	PAT	PLD

Figura 32: Fluxo de transporte apresentado através da ferramenta MPEG TS Utils.

Para o ambiente de verificação das fases seguintes de desenvolvimento, foi definido que os estímulos estariam contidos no mesmo TS produzido e respectivas respostas corretas (sequências binárias que produziram correta reprodução nas saídas dos decodificadores de áudio e vídeo) estarão em arquivos contendo situações que reproduzam da maneira mais fiel possível o mundo real e/ou as possíveis ocorrências de falhas que serão lidos pelo *testbench* implementado.

A verificação dos níveis seguintes ocorreu através da **comparação automatizada dos arquivos** criados pela aplicação aos correspondentes modelos dos arquivos de referência.

4 Arquitetura do Demultiplexador Proposto

Neste capítulo, é abordado o nível de arquitetura do demultiplexador proposto. Dessa forma, é possível apresentar os passos realizados a partir da especificação até a escolha dos blocos a serem implementados no próximo nível. Primeiramente, são apresentadas as questões referentes aos recursos e operações da arquitetura necessária a um demultiplexador. Posteriormente, é apresentada a microarquitetura selecionada para ser implementada.

4.1 Visão Comportamental no Nível Arquitetural do Demultiplexador

Como foi assumido anteriormente, a **especificação no nível de arquitetura** é dada por um conjunto de recursos funcionais, um conjunto de restrições e um grafo de sequenciamento. Aqui é apresentada tal especificação para o demultiplexador proposto.

4.1.1 Recursos

Aqui são apresentados os três tipos de recurso do demultiplexador.

Como **recursos funcionais**, o demultiplexador possui apenas os do tipo aplicação específica, são eles:

- Filtro de pacote TS;
 - Filtro de pacote de programa;
 - Analisador de cabeçalho de pacote TS;
 - Analisador de campo de adaptação;
 - Filtro de PCR;
 - Analisador de cabeçalho de pacote PES;
 - Filtro de PTS/DTS;
 - Analisador de cabeçalho de PSI/SI;
 - Filtro de seção PAT;
 - Filtro de seção PMT;
 - Filtro de seção SDT;
 - Filtros de seção CAT, NIT, BAT, EIT, RST, TDT, TOT, ST, PCAT, BIT, NBIT, LDT, LIT, ERT, ITT.
 - Detector de CRC-32 para pacotes PSI/SI.
-

Nota-se na lista acima uma distinção feita entre os filtros de seção PAT, PMT e SDT, dos demais. Para que um demultiplexador seja aderente ao SBTVD, todos os filtros de seção citados são necessários, como apresentado anteriormente na figura 29 da página 81. No quesito implementação, devido à complexidade do atendimento à especificação completa proposta no SBTVD, o objetivo deste trabalho restringiu-se a apresentar uma prova de conceito de concepção de IP de demultiplexador, percorrendo todas as etapas de projeto estruturado de circuitos integrados, utilizando um conjunto funcional mínimo – PAT e PMT – e um exemplo de seção SDT. A implementação dos demais filtros – CAT, NIT, BAT, EIT, RST, TDT, TOT, ST, PCAT, BIT, NBIT, LDT, LIT, ERT, ITT – embora façam parte do projeto principal demultiplexador em desenvolvimento no LASIC/UFPB, não estão contemplados neste trabalho.

Como **recursos de memória**, o demultiplexador possui:

- Fila de entrada de pacotes TS;
- Registrador de pacote TS;
- Fila de saída de dados de PCR;
- Fila de saída de dados de vídeo;
- Fila de saída de dados de áudio;
- Fila de saída de dados PES;
- Fila de saída de dados de PTS/DTS;
- Fila interna de dados PSI/SI para o detector de CRC-32;
- Fila de saída de dados PSI/SI.

Como **recursos de interface**, o demultiplexador possui dois tipos: *pad* de E/S e circuito de interfaceamento interno. Como *pad* de E/S, o demultiplexador possui:

- | | |
|-------------|------------|
| ● TS; | ● Áudio; |
| ● Programa; | ● Dados; |
| ● PCR; | ● PTS/DTS; |
| ● Vídeo; | ● PSI/SI. |

Como circuito de interfaceamento interno, o demultiplexador possui:

- Interfaceamento interno de Pacote válido;
- Interfaceamento interno de Pacote de programa;
- Interfaceamento interno de Cabeçalho;
- Interfaceamento interno de *Payload*;
- Interfaceamento interno de Tipo de *payload*;
- Interfaceamento interno de Campo PCR;
- Interfaceamento interno de *Payload* do tipo PES;
- Interfaceamento interno de *Payload* do tipo PSI/SI;
- Interfaceamento interno de Byte de Áudio;
- Interfaceamento interno de Byte de Vídeo;
- Interfaceamento interno de Byte de dados PES;
- Interfaceamento interno de Byte de PTS/DTS;
- Interfaceamento interno de Seção PAT;
- Interfaceamento interno de Seção PMT;
- Interfaceamento interno de Seção SDT;
- Interfaceamento interno de Seção PAT filtrada;
- Interfaceamento interno de Seção PMT filtrada;
- Interfaceamento interno de Seção SDT filtrada.

4.1.2 Restrições

A única **restrição de interface** proposta para este demultiplexador é relacionado ao formato de entrada que deve obedecer ao protocolo AMBA AXI (ARM IHI 0022B, 2004). Essa restrição se aplica às interfaces TS e Programa.

Quanto a **restrições de implementação**, por se tratar do desenvolvimento de um IP, não foi definida nenhuma restrição *a priori* em relação a área e desempenho.

4.1.3 Grafo de sequenciamento

Com o grafo de sequenciamento ordenado apresentado na figura 33, é possível

extrair as **operações** e **dependências** da arquitetura do demultiplexador, enquanto o seu **scheduling** é definido.

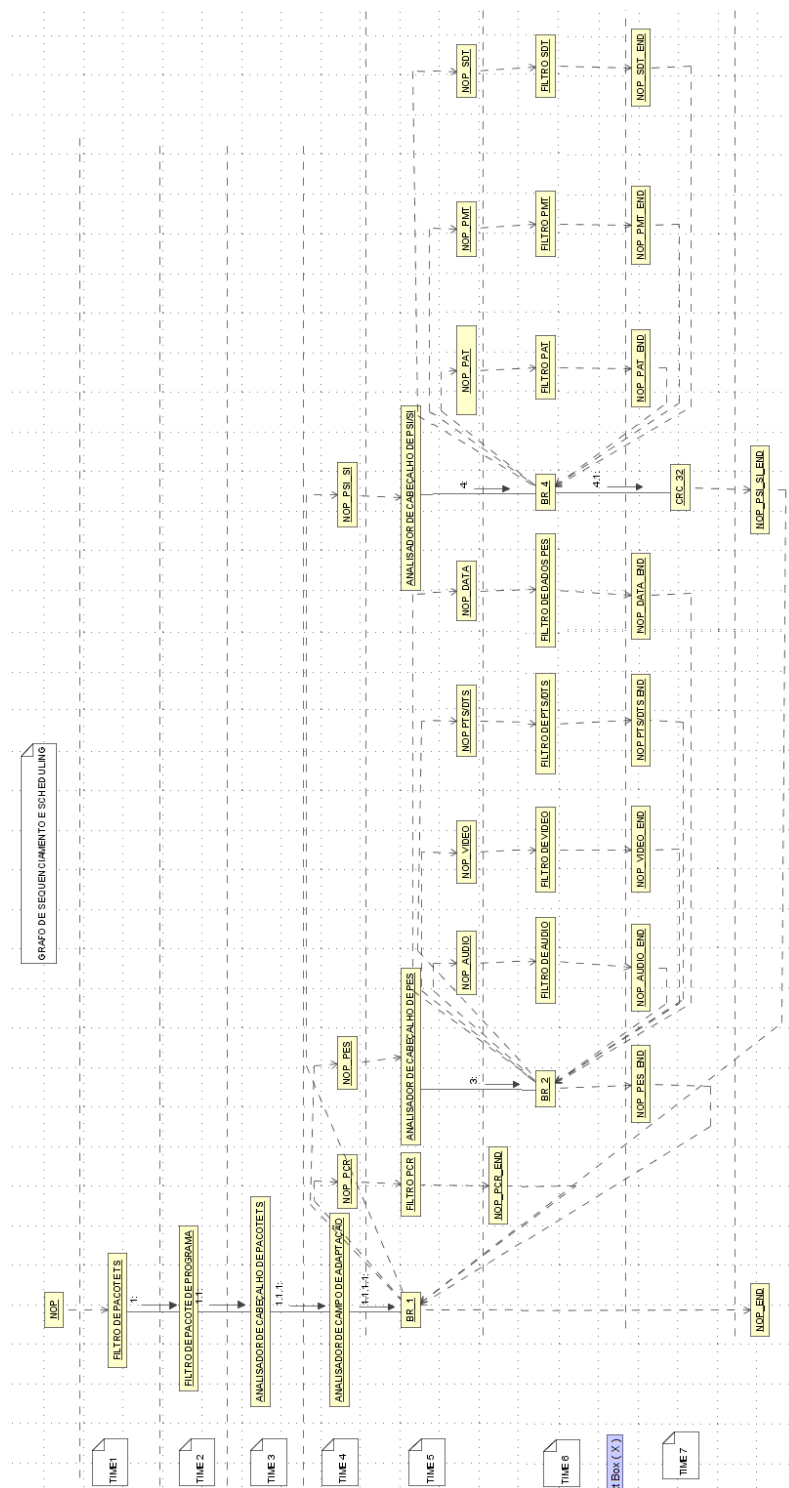


Figura 33: Gráfico de sequenciamento ordenado das operações do demultiplexador proposto.

4.1.4 Grafo de Fluxo de Dados

Com as operações posicionadas no tempo, fica mais fácil visualizar a questão da **disponibilidade de dados** existente no demultiplexador. Com o grafo do fluxo de dados apresentado na figura 34, estas dependências são mais facilmente visualizadas.

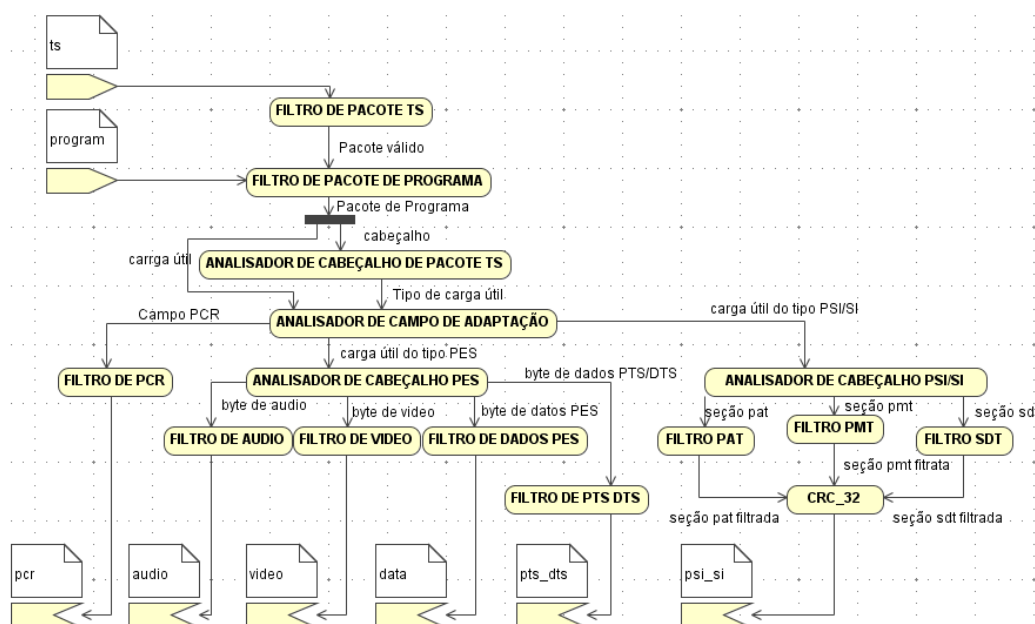


Figura 34: Grafo de fluxo de dados do Demultiplexador.

Em relação ao **posicionamento espacial**, foi definido o caso simples de *binding* para todo o demultiplexador. Todas as operações possuirão recurso dedicado, isto é, para cada operação, um recurso.

Aqui é feita uma escolha arquitetural paralelo *versus* serial para o demultiplexador, permitindo que as operações de filtro de PCR, assim como a análise gramatical de pacotes do tipo PES e do tipo PSI/SI, possam ser feitas paralelamente. A análise de mais de um pacote seria possível com a duplicação das operações de análise de cabeçalho de pacote TS e de análise de campo de adaptação. Mas, como o objetivo escolhido para o demultiplexador foi pela simplicidade, foi definido que tratamento do pacotes seria de um por vez.

Um conjunto de **sinais de ativação e conclusão** também foram definidos. Para cada operação existe apenas um sinal habilitador e uma *flag*. A *flag* serve para avisar que a operação foi concluída e é ligada ao sinal habilitador da operação seguinte. O conjunto

de sinais de ativação e conclusão existentes no módulo são identificados a seguir:

- *Flag* de filtro de pacote TS;
 - Habilitador de filtro de pacote de programa;
 - *Flag* de filtro de pacote de programa;
 - Habilitador de escrita na fila de entrada;
 - Habilitador de leitura na fila de entrada;
 - Habilitador de analisador de cabeçalho de pacote TS;
 - *Flag* de analisador de cabeçalho de pacote TS;
 - Habilitador de analisador de campo de adaptação;
 - *Flag* de analisador de campo de adaptação;
 - Habilitador de filtro de PCR;
 - *Flag* de filtro de PCR;
 - Habilitador de escrita na fila de saída de PCR;
 - Habilitador de leitura na fila de saída PCR;
 - Habilitador de analisador de cabeçalho de pacote PES;
 - *Flag* de analisador de cabeçalho de pacote PES;
 - Habilitador de escrita na fila de saída de vídeo;
 - Habilitador de leitura na fila de saída de vídeo;
 - Habilitador de escrita na fila de saída de áudio;
 - Habilitador de leitura na fila de saída de áudio;
 - Habilitador de escrita na fila de saída de dados PES;
 - Habilitador de leitura na fila de saída de dados PES;
 - Habilitador de escrita na fila de saída de PTS/DTS;
 - Habilitador de leitura na fila de saída de PTS/DTS;
 - Habilitador de analisador de cabeçalho de PSI/SI;
 - *Flag* de analisador de cabeçalho de PSI/SI;
 - Habilitador de filtro de seção PAT;
 - Habilitador de filtro de seção PMT;
 - Habilitador de filtro de seção SDT;
 - *Flag* de filtro de seção;
 - Habilitador de detector de CRC-32;
 - *Flag* de detector de CRC-32;
 - Habilitador de escrita na fila interna de CRC-32;
 - Habilitador de leitura na fila interna de CRC-32;
 - Habilitador de escrita na fila de saída de PSI/SI;
 - Habilitador de leitura na fila de saída de PSI/SI.
-

4.2 Microarquitetura do Demultiplexador

Várias escolhas referentes a compromissos arquiteturais podem ser feitas neste nível de projeto. No caso deste trabalho, buscou-se atender apenas as questões robustez funcional e modularidade/reuso de blocos/sub-blocos.

Assim, foi elaborado uma **visão estrutural da arquitetura** com interconexão de todos os recursos. A Figura 35 ilustra essa visão do demultiplexador. Posteriormente, são apresentadas as **decisões** tomadas para o demultiplexador proposto nessa dissertação.

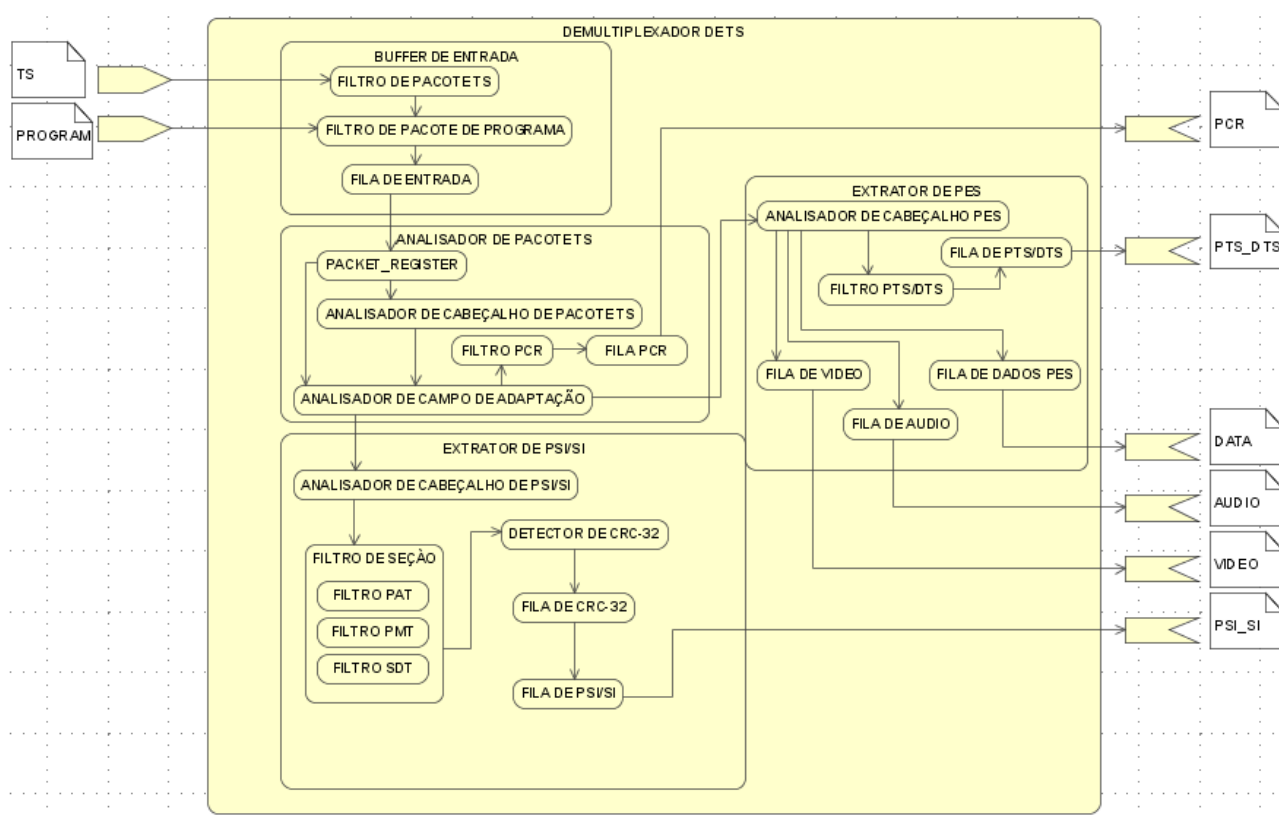


Figura 35: Microarquitetura do demultiplexador proposto.

Apenas um algoritmo foi escolhido por função. Devido à quantidade de algoritmos necessários para o demultiplexador, não foram feitas explorações de outras alternativas.

Os algoritmos poderiam ser particionados entre hardware e software. Optou-se por fazer em **hardware** devido aos objetivos iniciais dessa dissertação.

Todos os blocos funcionais do demultiplexador são implementadas através de **máquinas de estados finitas**.

Com o uso do protocolo AMBA AXI (ARM IHI 0022B, 2004), se torna necessário acrescentar **interfaces** de entrada e saída adicionais para as entradas TS e Programa. Para cada entrada, serão acrescentadas interfaces para o canal de endereço de escrita, canal de dados de escrita e canal de resposta de escrita. Esses canais realizam a transferência de dados de escrita em rajadas e utilizam o *handshake valid/ready* definido pelo protocolo.

Em relação à **memória**, são feitas escolhas relacionadas à estrutura da memória, largura de barramento versus frequência, e tipo de memória.

Quanto às estruturas da memória, foram utilizadas dois tipos de FIFO (*First In First Out*). Uma FIFO do tipo 4x188 bytes é dedicada aos pacotes TS de entrada. A largura de 188 bytes foi escolhida devido ao tamanho do pacote TS. Para as saídas de PCR, áudio, vídeo, dados, PTS/DTS e PSI/SI, as FIFOs são do tipo 256x1 bytes.

Toda a memória necessária é interna ao demultiplexador, e optou-se pelo modelo SRAM, pela facilidade de implementação e velocidade.

5 Implementação RTL e Mapeamento Tecnológico do Demultiplexador Proposto

Este capítulo apresenta a implementação RTL e mapeamento em FPGA do demultiplexador proposto assim como as verificações nos dois níveis. Com isso, será possível visualizar os detalhes de implementação que foram utilizados para sua descrição e detalhes do plano de verificação funcional, seguindo-se seu mapeamento tecnológico para prototipagem em FPGA.

O capítulo apresenta todos os blocos da **visão comportamental** no nível RTL do demultiplexador, que está dividido conforme implementação. Para cada bloco, quatro pontos serão descritos: **propósito**, **blocos instanciados**, **entradas principais** e **saídas principais**. Para o caso dos blocos onde foram desenvolvidas **máquinas de estados finitas**, uma figura é apresentada com os estados e transições presentes.

A **verificação funcional** no nível RTL foi realizada através da metodologia **Bottom-Up**, usando o método da **simulação**. Com o objetivo de verificar as características listadas no plano de verificação para cada bloco, foram construídos *testbenches* e vetores de teste. A **validação** foi feita através de um fluxo de transporte real. O TS foi colocado na especificação de alto-nível e na implementação RTL para que os arquivos de saída de ambos fossem comparados a fim de atestar a igualdade deles.

5.1 Visão Comportamental no Nível RTL

O desenvolvimento foi feito através da **linguagem de descrição de hardware** Verilog. A escolha da linguagem levou em consideração as ferramentas disponíveis para desenvolvimento no âmbito do LASIC/UFPB. Apoiado pela FINEP, através do projeto de softwares para universidades, o LASIC/UFPB disponibilizou o pacote do *High Education Program* – HEP – da empresa Mentor Graphics para auxiliar o desenvolvimento.

Durante a codificação, procurou-se utilizar boas práticas, como separar os blocos combinacionais dos blocos sequenciais na descrição de cada módulo. Foi utilizado um **estilo de codificação** que levou em conta questões como: escrita do código em um formato tabular, uso de endentação de código consistente com espaços, uso de uma

sentença Verilog por linha, uma declaração de porta por linha, preservação da ordem das portas, declaração de sinais internos. Além dessas questões, também é possível identificar uma convenção de nomeação e de comentários através de todos os blocos.

A seguir, são apresentados os blocos do demultiplexador desenvolvido.

- TS_DEMUX

Propósito: O TS_DEMUX tem a função de extrair de um TS os pacotes de vídeo, áudio, vídeo, *clock*, sincronização, dados privados e serviços de navegação (PSI/SI). E enviar esses pacotes para uma memória onde os outros componentes do decodificador MPEG-2 poderão acessá-los.

Blocos instanciados: IN_BUFFER, TRANSPORT_PACKET_ANALYSER, PES_EXTRACTOR e PSI_SI_EXTRACTOR.

Entradas principais: mpeg_transport_stream, program_pid e sinais do protocolo AMBA AXI.

Saídas principais: pcr_output, video_output, audio_output, data_output e psi_si_output.

5.1.1 IN_BUFFER

Propósito: O IN_BUFFER tem a função de detectar os pacotes TS e armazená-los em uma fila de entrada.

Blocos instanciados: TS_PACKET_FILTER, PROGRAM_PACKET_FILTER e IN_FIFO.

Entradas principais: mpeg_transport_stream, program_pid, pcr_pid, video_pid, audio_pid, data_pid e sinais do protocolo AMBA AXI.

Saída principal: transport_packet.

- TS_PACKET_FILTER

Propósito: O TS_PACKET_FILTER tem a função de detectar o byte de sincronização 0x47 do pacote TS e de fazer a conversão serial/paralelo dos 188 bytes de

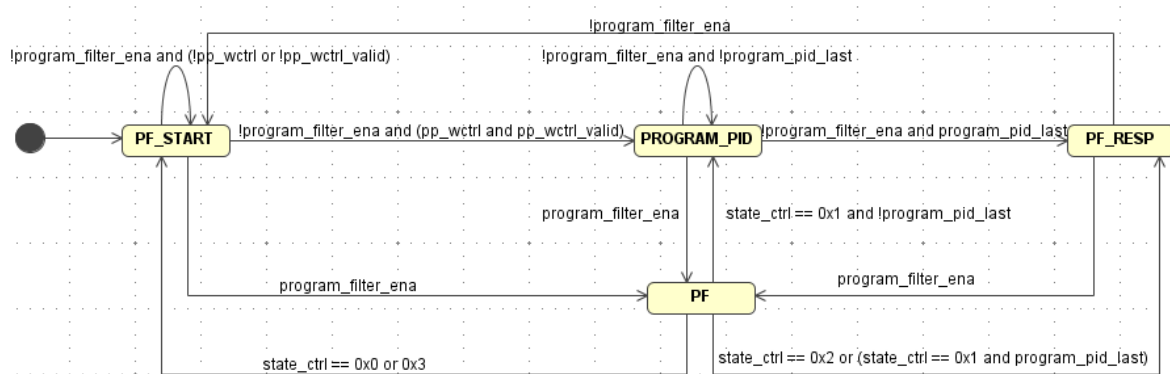


Figura 37: Estados e transições da máquina de estados finita do PROGRAM_PACKET_FILTER.

- IN_FIFO

Propósito: O IN_FIFO tem a função de atuar como uma fila síncrona para os pacotes TS de programa e de avisar quando está cheia ou vazia.

Blocos instanciados: IN_MEM.

Entrada principal: wr_data.

Saída principal: rd_data.

- IN_MEM

Propósito: O IN_MEM tem a função de atuar como a memória propriamente dita da fila síncrona IN_FIFO e possui ponteiros e *flags* para escrita e leitura.

Blocos instanciados: nenhum.

Entradas principais: w_data, w_addr e r_addr.

Saída principal: r_data.

5.1.2 TRANSPORT_PACKET_ANALYSER

Propósito: O TRANSPORT_PACKET_ANALYSER tem a função de analisar o cabeçalho de pacote TS e o campo de adaptação, e extrair os dados de sincronização PCR. Os estados e transições desse bloco são apresentados na figura 38.

Blocos instanciados: TRANSPORT_PACKET_HEADER_ANALYSER,

ADAPTATION_FIELD_ANALYSER, PCR_FILTER e PCR_FIFO. Possui o PACKET_REGISTER.

Entradas principais: transport_packet, pcr_pid, video_pid, audio_pid e data_pid.

Saídas principais: pcr_output, pid e payload.

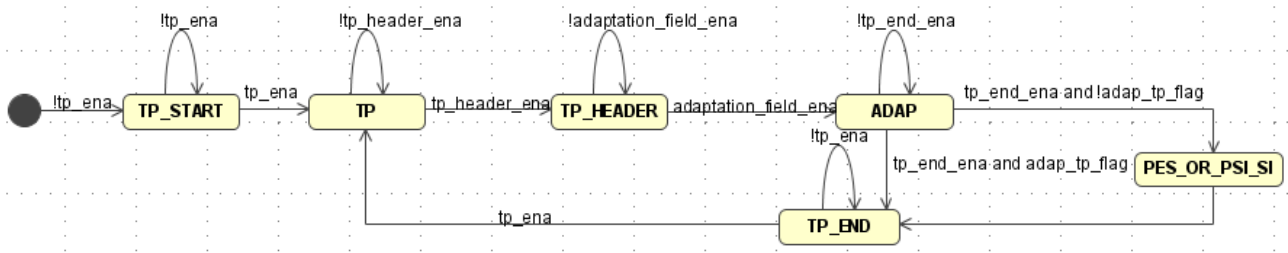


Figura 38: Estados e transições da máquina de estados finita do TRANSPORT_PACKET_ANALYSER.

- TRANSPORT_PACKET_HEADER_ANALYSER

Propósito: O TRANSPORT_PACKET_HEADER_ANALYSER tem a função de analisar o cabeçalho de pacote TS. Os estados e transições desse bloco são apresentados na figura 39.

Blocos instanciados: nenhum.

Entradas principais: transport_packet_header, pcr_pid, video_pid, audio_pid e data_pid.

Saídas principais: pid e payload_type.

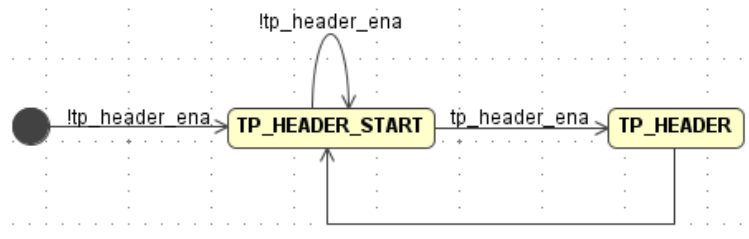


Figura 39: Estados e transições da máquina de estados finita do TRANSPORT_PACKET_HEADER_ANALYSER.

- ADAPTATION_FIELD_ANALYSER

Propósito: O ADAPTATION_FIELD_ANALYSER tem a função de analisar o campo de adaptação e extrair os dados de sincronização PCR. Os estados e transições desse bloco são apresentados na figura 40.

Blocos instanciados: nenhum.

Entrada principal: transport_packet_payload.

Saídas principais: pcr_field e payload.

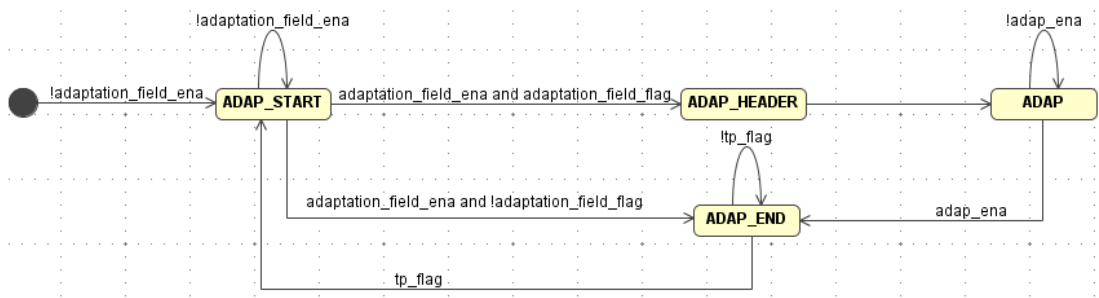


Figura 40: Estados e transições da máquina de estados finita do ADAPTATION_FIELD_ANALYSER.

- PCR_FILTER

Propósito: O PCR_FILTER tem a função de fazer a conversão paralelo/serial dos dados de recuperação de relógio PCR. Os estados e transições desse bloco são apresentados na figura 41.

Blocos instanciados: nenhum.

Entrada principal: pcr_field.

Saída principal: pcr_byte.

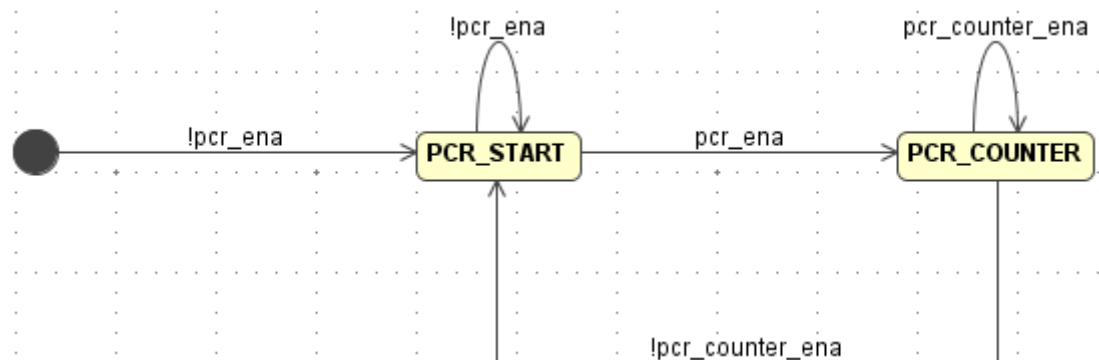


Figura 41: Estados e transições da máquina de estados finita do PCR_FILTER.

- PCR_FIFO

Propósito: O PCR_FIFO tem a função de atuar como uma fila síncrona para os dados de recuperação de relógio PCR e de avisar quando está cheia ou vazia.

Blocos instanciados: PCR_MEM.

Entrada principal: wr_data.

Saída principal: rd_data.

- PCR_MEM

Propósito: O PCR_MEM tem a função de atuar como a memória propriamente dita da fila síncrona PCR_FIFO e possui ponteiros e *flags* para escrita e leitura.

Blocos instanciados: nenhum.

Entradas principais: w_data, w_addr e r_addr.

Saída principal: r_data.

5.1.3 PES_EXTRACTOR

Propósito: O PES_EXTRACTOR tem a função de analisar o cabeçalho de pacote PES e extrair os dados de vídeo, dados de áudio e dados do pacote PES.

Blocos instanciados: PES_HEADER_ANALYSER, PTS_DTS_FILTER, VIDEO_FIFO, AUDIO_FIFO, DATA_FIFO e PTS_DTS_FIFO..

Entradas principais: payload, pid, video_pid, audio_pid e data_pid.

Saídas principais: video_output, audio_output e data_output.

- PES_HEADER_ANALYSER

Propósito: O PES_HEADER_ANALYSER tem a função de analisar o cabeçalho de pacote PES e extrair bytes referentes a dados de vídeo, dados de áudio e dados do pacote PES. Os estados e transições desse bloco são apresentados na figura 42.

Blocos instanciados: nenhum.

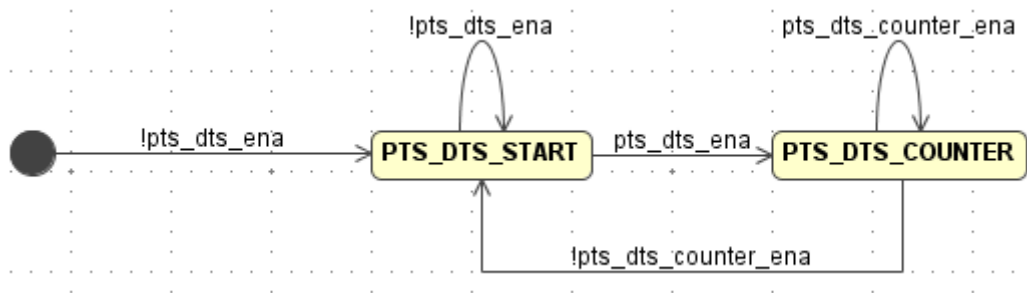


Figura 43: Estados e transições da máquina de estados finita do PTS_DTS_FILTER.

- PTS_DTS_FIFO

Propósito: O PTS_DTS_FIFO tem a função de atuar como uma fila síncrona para os dados de sincronização PTS/DTS e de avisar quando está cheia ou vazia.

Blocos instanciados: PTS_DTS_MEM.

Entrada principal: wr_data.

Saída principal: rd_data.

- PTS_DTS_MEM

Propósito: O PTS_DTS_MEM tem a função de atuar como a memória propriamente dita da fila síncrona PTS_DTS_FIFO e possui ponteiros e *flags* para escrita e leitura.

Blocos instanciados: nenhum.

Entradas principais: w_data, w_addr e r_addr.

Saída principal: r_data.

- VIDEO_FIFO

Propósito: O VIDEO_FIFO tem a função de atuar como uma fila síncrona para os dados de vídeo e de avisar quando está cheia ou vazia.

Blocos instanciados: VIDEO_MEM.

Entrada principal: wr_data.

Saída principal: rd_data.

- VIDEO_MEM

Propósito: O VIDEO_MEM tem a função de atuar como a memória propriamente dita da fila síncrona VIDEO_FIFO e possui ponteiros e *flags* para escrita e leitura.

Blocos instanciados: nenhum.

Entradas principais: w_data, w_addr e r_addr.

Saída principal: r_data.

- AUDIO_FIFO

Propósito: O AUDIO_FIFO tem a função de atuar como uma fila síncrona para os dados de áudio e de avisar quando está cheia ou vazia.

Blocos instanciados: AUDIO_MEM.

Entrada principal: wr_data.

Saída principal: rd_data.

- AUDIO_MEM

Propósito: O AUDIO_MEM tem a função de atuar como a memória propriamente dita da fila síncrona AUDIO_FIFO e possui ponteiros e *flags* para escrita e leitura.

Blocos instanciados: nenhum.

Entradas principais: w_data, w_addr e r_addr.

Saída principal: r_data.

- DATA_FIFO

Propósito: O DATA_FIFO tem a função de atuar como uma fila síncrona para os dados do pacote PES e de avisar quando está cheia ou vazia.

Blocos instanciados: DATA_MEM.

Entrada principal: wr_data.

Saída principal: rd_data.

- DATA_MEM

Propósito: O DATA_MEM tem a função de atuar como a memória propriamente dita da fila síncrona DATA_FIFO e possui ponteiros e *flags* para escrita e leitura.

Blocos instanciados: nenhum.

Entradas principais: w_data, w_addr e r_addr.

Saída principal: r_data.

5.1.4 PSI_SI_EXTRACTOR

Propósito: O PSI_SI_EXTRACTOR tem a função de analisar as seções PSI/SI, verificar o CRC-32 e extrair os dados das seções PSI/SI, e extrair os PIDs de vídeo, de áudio e de dados de pacote PES. Os estados e transições desse bloco são apresentados na figura 44.

Blocos instanciados: PSI_SI_HEADER_ANALYSER, SECTION_FILTER, CRC_32_DETECTOR, CRC_32_FIFO e PSI_SI_FIFO.

Entrada principal: payload.

Saídas principais: psi_si_output, pcr_pid, video_pid, audio_pid e data_pid.

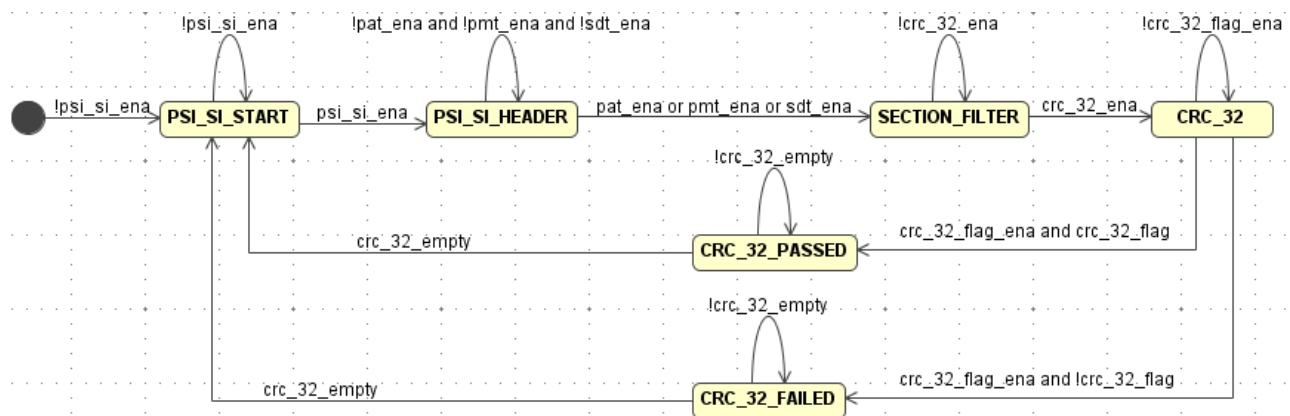


Figura 44: Estados e transições da máquina de estados finita do PSI_SI.

- PSI_SI_HEADER_ANALYSER

Propósito: O PSI_SI_HEADER_ANALYSER tem a função de analisar o cabeçalho das seções PSI/SI e definir o tipo de seção. Os estados e transições desse bloco são apresentados na figura 45.

Blocos instanciados: nenhum.

Entrada principal: payload.

Saída principal: psi_si_payload.

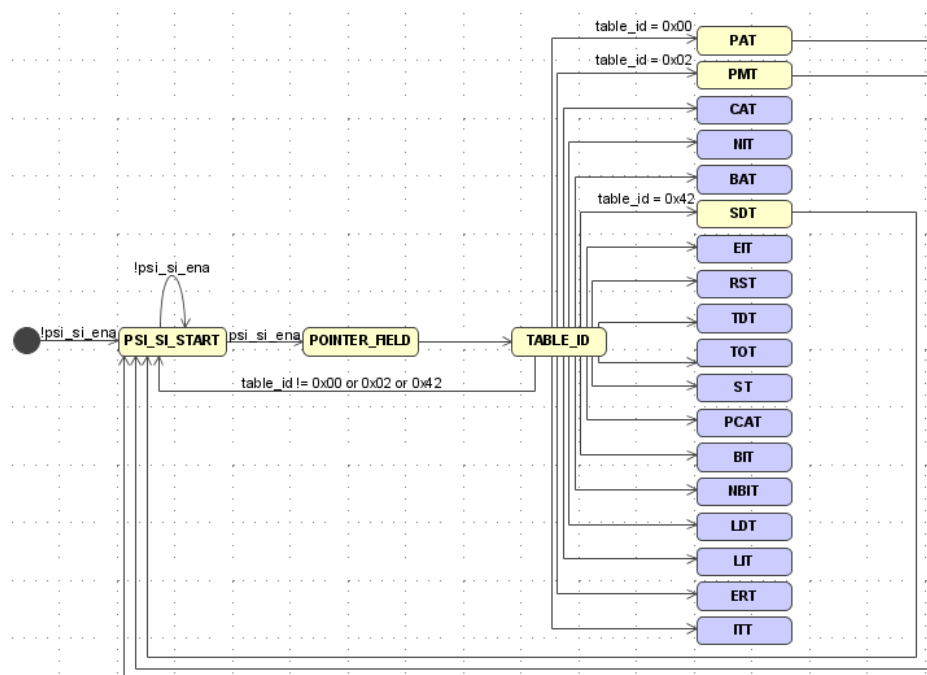


Figura 45: Estados e transições da máquina de estados finita do PSI_SI_HEADER_ANALYSER.

Na figura 45, os estados da cor azul não foram implementados, e servem apenas de referência para posterior implementação.

- SECTION_FILTER

Propósito: O SECTION_FILTER tem a função de filtrar as seções PSI/SI e extrair informações necessárias à de-multiplexagem.

Blocos instanciados: PAT_FILTER, PMT_FILTER e SDT_FILTER.

Entrada principal: psi_si_payload.

Saídas principais: section_filter_data_byte, pcr_pid, video_pid, audio_pid e

data_pid.

- PAT_FILTER

Propósito: O PAT_FILTER tem a função de filtrar a seção PAT. Os estados e transições desse bloco são apresentados na figura 46.

Blocos instanciados: nenhum.

Entrada principal: psi_si_payload.

Saída principal: pat_data_byte.

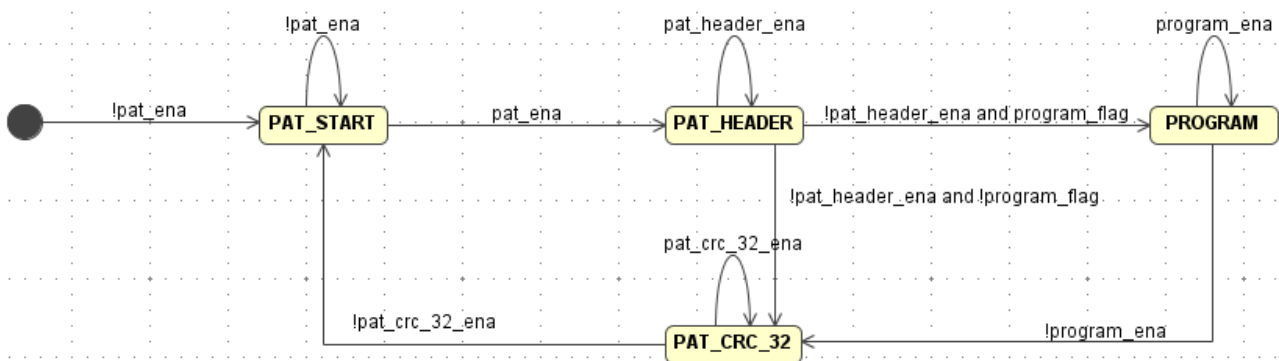


Figura 46: Estados e transições da máquina de estados finita do PAT_FILTER.

- PMT_FILTER

Propósito: O PMT_FILTER tem a função de filtrar a seção PMT. Os estados e transições desse bloco são apresentados na figura 47.

Blocos instanciados: nenhum.

Entrada principal: psi_si_payload.

Saídas principais: pmt_data_byte, pcr_pid, video_pid, audio_pid e data_pid.

da seção. Os estados e transições desse bloco são apresentados na figura 49.

Blocos instanciados: nenhum.

Entradas principais: d.

Saídas principais: d_out e crc_32_flag.

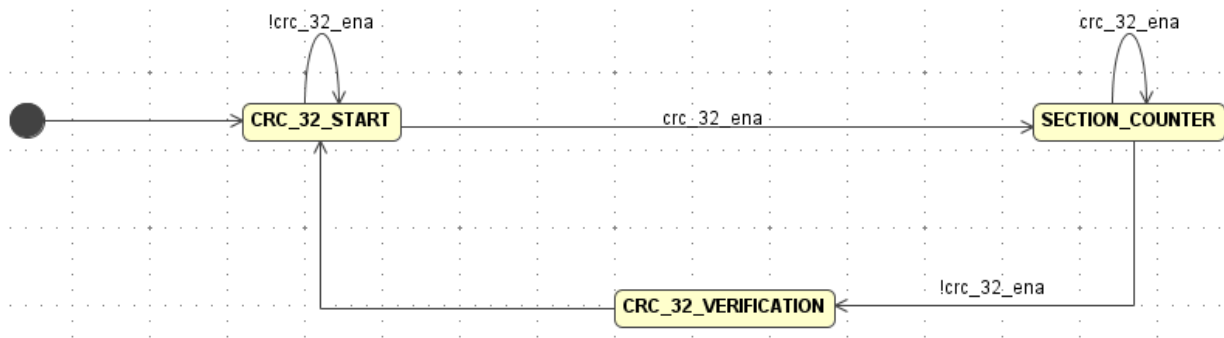


Figura 49: Estados e transições da máquina de estados finita do CRC_32_DETECTOR.

- CRC_32_FIFO

Propósito: O CRC_32_FIFO tem a função de atuar como uma fila síncrona para os dados do serviço de navegação (PSI/SI) que ainda não tiveram o CRC verificado e de avisar quando está cheia ou vazia.

Blocos instanciados: CRC_32_MEM.

Entrada principal: wr_data.

Saída principal: rd_data.

- CRC_32_MEM

Propósito: O CRC_32_MEM tem a função de atuar como a memória propriamente dita da fila síncrona CRC_32_FIFO e possui ponteiros e *flags* para escrita e leitura.

Blocos instanciados: nenhum.

Entradas principais: w_data, w_addr e r_addr.

Saída principal: r_data.

- PSI_SI_FIFO

Propósito: O PSI_SI_FIFO tem a função de atuar como uma fila síncrona para os dados do serviço de navegação (PSI/SI) e de avisar quando está cheia ou vazia.

Blocos instanciados: PSI_SI_MEM.

Entrada principal: wr_data.

Saída principal: rd_data.

- PSI_SI_MEM

Propósito: O PSI_SI_MEM tem a função de atuar como a memória propriamente dita da fila síncrona PSI_SI_FIFO e possui ponteiros e *flags* para escrita e leitura.

Blocos instanciados: nenhum.

Entradas principais: w_data, w_addr e r_addr.

Saída principal: r_data.

5.2 Verificação Funcional do Demultiplexador

A verificação funcional do demultiplexador teve início a partir da aplicação do **plano de verificação** elaborado a partir do que foi detalhado no capítulo 2, mais especificamente no ponto 2.1.8. Nesta seção, são detalhados os pontos desse plano, a saber: montagem da tabela com os **componentes de verificação**, detalhamento das **características** a serem verificadas por grande bloco do sistema, **requisitos de cobertura e de conclusão**, bem como os **recursos utilizados**.

A tabela 4 apresenta os componentes de verificação funcional que compreendem: a metodologia (*Bottom-Up*) e o método de verificação (Simulação), este último composto de ferramenta, linguagem e arquivos a serem utilizados (*testbench*).

O detalhamento das características verificáveis por grande bloco é apresentado na sequência, em 5.2.1, através de tabelas referenciando características e respectivos arquivos de *testbench*.

Em 5.2.2 são detalhados os requisitos de cobertura e conclusão da verificação e, em 5.2.3 os recursos utilizados.

Tabela 4: Componentes do plano de verificação funcional do demultiplexador proposto.

Componente	Descrição
Metodologia de Verificação	Bottom-Up
Método de Verificação	Simulação
Componentes de Simulação	
Ferramenta	QuestaSim 6.4
Linguagem	Verilog 2001, SystemVerilog
Arquivos de Testbench	tb_in_fifo.sv tb_in_mem.sv tb_program_packet_filter.sv tb_ts_packet_filter.sv tb_in_buffer.sv tb_adaptation_field_analyser.sv tb_pcr_filter.sv tb_pcr_fifo.sv tb_pcr_mem.sv tb_transport_packet_header_analyser.sv tb_transport_packet_extractor.sv tb_audio_fifo.sv tb_audio_mem.sv tb_data_fifo.sv tb_data_mem.sv tb_video_fifo.sv tb_video_mem.sv tb_pts_dts_fifo.sv tb_pts_dts_mem.sv tb_pts_dts_filter.sv tb_pes_header_analyser.sv tb_pes_extractor.sv tb_crc_32_detector.sv tb_crc_32_fifo.sv tb_crc_32_mem.sv tb_psi_si_fifo.sv tb_psi_si_header_analyser.sv tb_psi_si_mem.sv tb_pat_filter.sv tb_pmt_filter.sv tb_sdt_filter.sv tb_section_filter.sv tb_psi_si_extractor.sv tb_ts_demux.sv

5.2.1 Características Verificadas por Bloco

Tendo o modelo de mais alto nível (em C/C++) sido validado com fluxos de referência conforme descrito no capítulo 3, as sequências binárias saídas dessa fase passaram ao papel de referências para as fases seguintes.

Devido à sua complexidade, o sistema alvo (demultiplexador) foi implementado em linguagem de descrição de hardware a partir de seus blocos mais elementares (*Bottom-Up*).

Para cada bloco, foram construídos *testbenches* (são os arquivos mencionados na tabela 4) pelo desmembramento de trechos dos fluxos binários de referência correspondentes a cada uma de suas funcionalidades.

A seguir, são detalhadas as **características verificadas** de cada bloco.

Para cada bloco, são apresentadas tabelas (de 5 a 18) contendo características identificadas por mnemônicos formados pelo [nome do bloco.Fn] com n sendo o número de diferentes características.

A tabela 5 apresenta as características verificadas para o bloco *ts_packet_filter*. O *tb_ts_packet_filter.sv* é arquivo de teste onde estão presentes tais características.

Tabela 5: Características verificadas do bloco ts_packet_filter.

Característica	Descrição
ts_packet_filter.F1	Controle de Escrita – A saída wctrl_ready deve ser 1 se a entrada wctrl_valid for para 1.
ts_packet_filter.F2	Byte de Entrada – A saída byte_in_ready deve ser 1 se a entrada byte_in_valid for para 1.
ts_packet_filter.F3	Resposta – As saídas resp e resp_valid devem ser 1 se a entrada byte_in_last for para 1.
ts_packet_filter.F4	SLR – Deslocamento para esquerda se byte_in_ready for 1.
ts_packet_filter.F5	Valid Packet – valid_packet deve manter 0x47 no primeiro byte.
ts_packet_filter.F6	Habilitador de Filtro de Programa – program_filter_ena deve ser 1 quando o primeiro byte e o byte 189 forem 0x47.
ts_packet_filter.F7	Reset – Se rst_n for ativado a qualquer tempo, deve voltar de maneira síncrona para o estado inicial SYNC_START.

A tabela 6 apresenta as características verificadas para o bloco *program_packet_filter*. O *tb_program_packet_filter.sv* é arquivo de teste onde estão presentes tais características.

Tabela 6: Características verificadas do bloco program_packet_filter.

Característica	Descrição
program_packet_filter.F1	Filtro de Programa – Se program_filter_ena for 1 e in_full for 0 e os 13 bits que representam o PID do pacote válido for igual a zero (PAT), ao program_pid, ao pcr_pid, ao video_pid, ao audio_pid ou ao data_pid, o valid_packet deve ir para o program_packet.
program_packet_filter.F2	Habilitador de Filtro de Programa – Se program_filter_ena for 0, program_packet se mantém
program_packet_filter.F3	FIFO de Entrada Cheio – Se in_full for 1, program_packet se mantém

A tabela 7 apresenta as características verificadas para o bloco *in_fifo* e demais blocos do tipo *fifo* (*pcr_fifo*, *audio_fifo*, *data_fifo*, *video_fifo*, *pts_dts_fifo*, *crc_32_fifo*, *psi_si_fifo*). Os arquivos de teste *tb_in_fifo.sv*, *tb_pcr_fifo.sv*, *tb_audio_fifo*, *tb_data_fifo*, *tb_video_fifo*, *tb_pts_dts_fifo*, *tb_crc_32_fifo*, *tb_psi_si_fifo* são os onde estão presentes tais características.

Tabela 7: Características verificadas do bloco *in_fifo*.

Característica	Descrição
in_fifo.F1	Apontador de Escrita - Se entrada <i>wr_ena</i> for 1 e a flag <i>full</i> for 0, próximo apontador de escrita deve ser o atual apontador de escrita mais 1.
in_fifo.F2	Apontador de Leitura - Se entrada <i>rd_ena</i> for 1 e a flag <i>empty</i> for 0, próximo apontador de leitura deve ser o atual apontador de leitura mais 1.
in_fifo.F1	Apontador de Escrita para última posição da FIFO – Se entrada <i>wr_ena</i> for 1 e a flag <i>full</i> for 0, próximo apontador de escrita deve ser 0.
in_fifo.F2	Apontador de Leitura para última posição da FIFO – Se entrada <i>rd_ena</i> for 1 e a flag <i>empty</i> for 0, próximo apontador de leitura deve ser 0.
in_fifo.F3	Flag de FIFO cheia – A saída <i>full</i> deve ser 1 quando a entrada <i>wr_ena</i> for 1, a entrada <i>rd_ena</i> for 0 e o próximo apontador de escrita for igual ao atual apontador de leitura.
in_fifo.F4	Flag de FIFO vazia – A saída <i>empty</i> deve ser 1 quando a entrada <i>rd_ena</i> for 1, a entrada <i>wr_ena</i> for 0 e o próximo apontador de leitura for igual ao atual apontador de escrita.

A tabela 8 apresenta as características verificadas para o bloco *in_mem* e demais blocos do tipo *mem* (*pcr_mem*, *audio_mem*, *data_mem*, *video_mem*, *pts_dts_mem*, *crc_32_mem*, *psi_si_mem*). Os arquivos de teste *tb_in_mem.sv*, *tb_pcr_mem.sv*, *tb_audio_mem*, *tb_data_mem*, *tb_video_mem*, *tb_pts_dts_mem*, *tb_crc_32_mem*, *tb_psi_si_mem* são os onde estão presentes tais características.

Tabela 8: Características verificadas do bloco *in_mem*.

Característica	Descrição
in_mem.F1	Leitura de dados – <i>r_data</i> deve ser o conteúdo da posição de memória <i>r_addr</i> .
in_mem.F2	Escrita de dados – Se <i>w_ena</i> for 1, a posição de memória <i>w_addr</i> deve receber <i>w_data</i> .

A tabela 9 apresenta as características verificadas para o bloco *transport_packet_header_analyser*. O *tb_transport_packet_header_analyser.sv* é arquivo de teste onde estão presentes tais características.

Tabela 9: Características verificadas do bloco *transport_packet_header_analyser*.

Característica	Descrição
transport_packet_header_analyser.F1	FSM – As transições e saídas da máquina de estados finita devem funcionar como o especificado.
transport_packet_header_analyser.F2	PID – A saída pid deve ser igual ao campo PID do pacote TS.
transport_packet_header_analyser.F3	Flag do Campo de Adaptação – A saída adaptation_field_flag deve ser 1 quando o campo adaptation field control do pacote TS for 0x2.

A tabela 10 apresenta as características verificadas para o bloco *adaptation_field_analyser*. O *tb_adaptation_field_analyser.sv* é arquivo de teste onde estão presentes tais características.

Tabela 10: Características verificadas do bloco *adaptation_field_analyser*.

Característica	Descrição
adaptation_field_analyser.F1	FSM – As transições e saídas da máquina de estados finita devem funcionar como o especificado.
adaptation_field_analyser.F2	PCR – Se adaptation_field_ena e adaptation_field_flag forem 1, a saída pcr_field deve ser igual ao campo pcr field do campo de adaptação.

A tabela 11 apresenta a característica verificada para o bloco *pcr_filter*. O *tb_pcr_filter.sv* é arquivo de teste onde está presente tal característica.

Tabela 11: Características verificadas do bloco *pcr_filter*.

Característica	Descrição
pcr_filter.F1	FSM – As transições e saídas da máquina de estados finita devem funcionar como o especificado.

A tabela 12 apresenta as características verificadas para o bloco *pes_header_analyser*. O *tb_pes_header_analyser.sv* é arquivo de teste onde estão presentes tais características.

Tabela 12: Características verificadas do bloco *pes_header_analyser*.

Característica	Descrição
pes_header_analyser.F1	FSM – As transições e saídas da máquina de estados finita devem funcionar como o especificado.
pes_header_analyser.F2	Packet Start Code Prefix – Se o campo Packet Start Code Prefix for 0x000001, o sinal interno pes_type deve ir para 1.
pes_header_analyser.F3	Contador PES – O pes_byte_counter não deve ultrapassar o valor 184 referente à <i>payload</i> do pacote TS.
pes_header_analyser.F4	PES Packet Data Byte – Para o caso do estado ser PES_PAYLOAD ou DATA_BYTE. Se o pid for igual ao video_pid, video_ena deve ir para 1. Se o pid for igual ao audio_pid, audio_ena deve ir para 1. Se o pid for igual ao data_pid, data_ena deve ir para 1.

A tabela 13 apresenta a característica verificada para o bloco *pts_dts_filter*. O *tb_pts_dts_filter.sv* é arquivo de teste onde está presente tal característica.

Tabela 13: Características verificadas do bloco *pts_dts_filter*.

Característica	Descrição
pts_dts_filter.F1	FSM – As transições e saídas da máquina de estados finita devem funcionar como o especificado.

A tabela 14 apresenta as características verificadas para o bloco *psi_si_header_analyser*. O *tb_psi_si_header_analyser.sv* é arquivo de teste onde estão presentes tais características.

Tabela 14: Características verificadas do bloco *psi_si_header_analyser*.

Característica	Descrição
psi_si_header_analyser.F1	FSM – As transições e saídas da máquina de estados finita devem funcionar como o especificado.
psi_si_header_analyser.F2	Table id – Se o campo table id for 0x00, pat_ena deve ir para 1. Se o campo table id for 0x02, pmt_ena deve ir para 1. Se o campo table id for 0x42, sdt_ena deve ir para 1.
psi_si_header_analyser.F3	Pointer Field – Deslocamento para esquerda se pointer_field for 1.

A tabela 15 apresenta as características verificadas para o bloco *pat_filter*. O *tb_pat_filter.sv* é arquivo de teste onde estão presentes tais características.

Tabela 15: Características verificadas do bloco *pat_filter*.

Característica	Descrição
pat_filter.F1	FSM – As transições e saídas da máquina de estados finita devem funcionar como o especificado.
pat_filter.F2	Contador PAT – O pat_byte_counter não deve ultrapassar o valor section_length referente ao tamanho da seção PAT.

A tabela 16 apresenta as características verificadas para o bloco *pmt_filter*. O *tb_pmt_filter.sv* é arquivo de teste onde estão presentes tais características.

Tabela 16: Características verificadas do bloco pmt_filter.

Característica	Descrição
pmt_filter.F1	FSM – As transições e saídas da máquina de estados finita devem funcionar como o especificado.
pmt_filter.F2	Contador PMT – O pmt_byte_counter não deve ultrapassar o valor section_length referente ao tamanho da seção PMT.
pmt_filter.F3	PCR PID – Para o caso do estado ser PMT_HEADER. A saída pcr_pid deve corresponder ao campo PCR PID presente na seção PMT.
pmt_filter.F4	PID – Para o caso do estado ser ELEMENTARY_PID. A saída video_pid deve corresponder ao campo elementary PID, se o campo stream type for 0x02 ou 0x1b. A saída audio_pid deve corresponder ao campo elementary PID, se o campo stream type for 0x03 ou 0x11.

A tabela 17 apresenta as características verificadas para o bloco *sdt_filter*. O *tb_sdt_filter.sv* é arquivo de teste onde estão presentes tais características.

Tabela 17: Características verificadas do bloco sdt_filter.

Característica	Descrição
sdt_filter.F1	FSM – As transições e saídas da máquina de estados finita devem funcionar como o especificado.
sdt_filter.F2	Contador PAT – O pat_byte_counter não deve ultrapassar o valor section_length referente ao tamanho da seção PAT.

A tabela 18 apresenta as características verificadas para o bloco *crc_32_detector*. O *tb_crc_32_detector.sv* é arquivo de teste onde estão presentes tais características.

Tabela 18: Características verificadas do bloco crc_32_detector.

Característica	Descrição
crc_32_detector.F1	FSM – As transições e saídas da máquina de estados finita devem funcionar como o especificado.
crc_32_detector.F2	Verificação CRC-32 – Para o caso da transição do estado SECTION_COUNTER para CRC_32_VERIFICATION. Se crc_32_value for 0x00000000, a saída crc_32_flag deve ir para 1, senão deve ir para 0.

5.2.2 Requisitos de Cobertura e de Conclusão da Verificação

Nunca é possível determinar todos os erros que serão encontrados em um projeto. Para estabelecer um compromisso, foi preciso definir critérios que, uma vez alcançados,

significam o fim da etapa de verificação. Definiram-se, como apresenta a tabela 19, os **critérios de conclusão** da etapa de verificação.

Tabela 19: Critérios de conclusão da verificação funcional do demultiplexador.

Critério	Nível necessário	Modo
Deteção de erros diários para o projeto do demultiplexador	Menos de 1 erro/dia	Teste de Verificação Contínua
Cobertura funcional dos blocos	100%	Testbench de Verificação

A cobertura funcional dos blocos foi um dos critérios estabelecidos para a conclusão da verificação funcional. Como exemplo do que foi realizado em cada bloco, a cobertura funcional realizada no bloco *program_filter* é apresentada na figura 50.

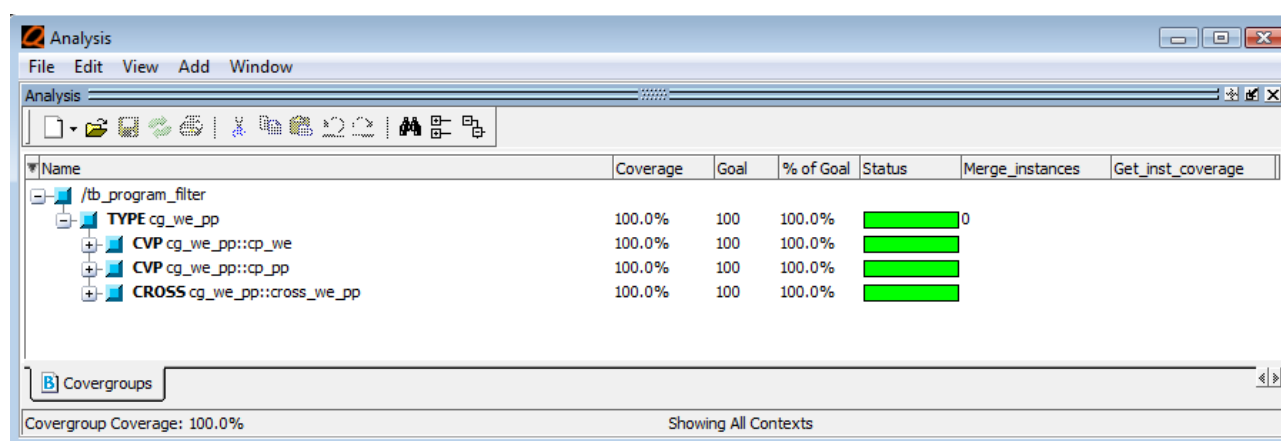


Figura 50: Exemplo de cobertura funcional realizada nos blocos.

5.2.3 Recursos Utilizados

Para cada tarefa de verificação, foi necessário definir **recursos** a serem utilizados. Para o planejamento da etapa de verificação, foi considerado a realidade do ambiente em que foi realizado o projeto. A tabela 20 apresenta os utilizados durante o projeto do demultiplexador.

Tabela 20: Recursos utilizados na verificação funcional do demultiplexador.

Recurso	Quantidade	Descrição	Duração
Recursos de Engenharia			
Engenheiro de Verificação Junior	1	Pequena Experiência	4 meses
Recursos Computacionais			
Sistema/Unidade Servidor para Verificação	1	Dual Core CPU, 1 GB RAM	4 meses
Recursos de Software/Licença			
Mentor Graphics QuestaSim	1	Version 6.4	4 meses

5.3 Validação no nível RTL

A **validação** foi feita através de um *testbench* do tipo comparação de saídas. Foi utilizado o fluxo de transporte de referência gerado pela ferramenta TSReader v2.6.41. Esse TS foi aplicado à especificação de alto-nível, como apresentado na figura 30 da página 83 e à implementação RTL.

O arquivo de teste *tb_ts_demux.sv* foi utilizado para simular a aplicação do fluxo de transporte de referência e gerar os arquivos de saída de vídeo e de áudio correspondentes à implementação RTL.

Os arquivos de saída da especificação de alto-nível (devidamente validados pela reprodução correta de seus conteúdos demultiplexados) e da simulação da implementação RTL foram comparados pela ferramenta HDD Hex Editor Neo v4.93 – figuras 51 e 52 – usando a opção do método de comparação de arquivos que destaca a diferença entre eles. Assim, foi possível atestar a igualdade de ambos arquivos.

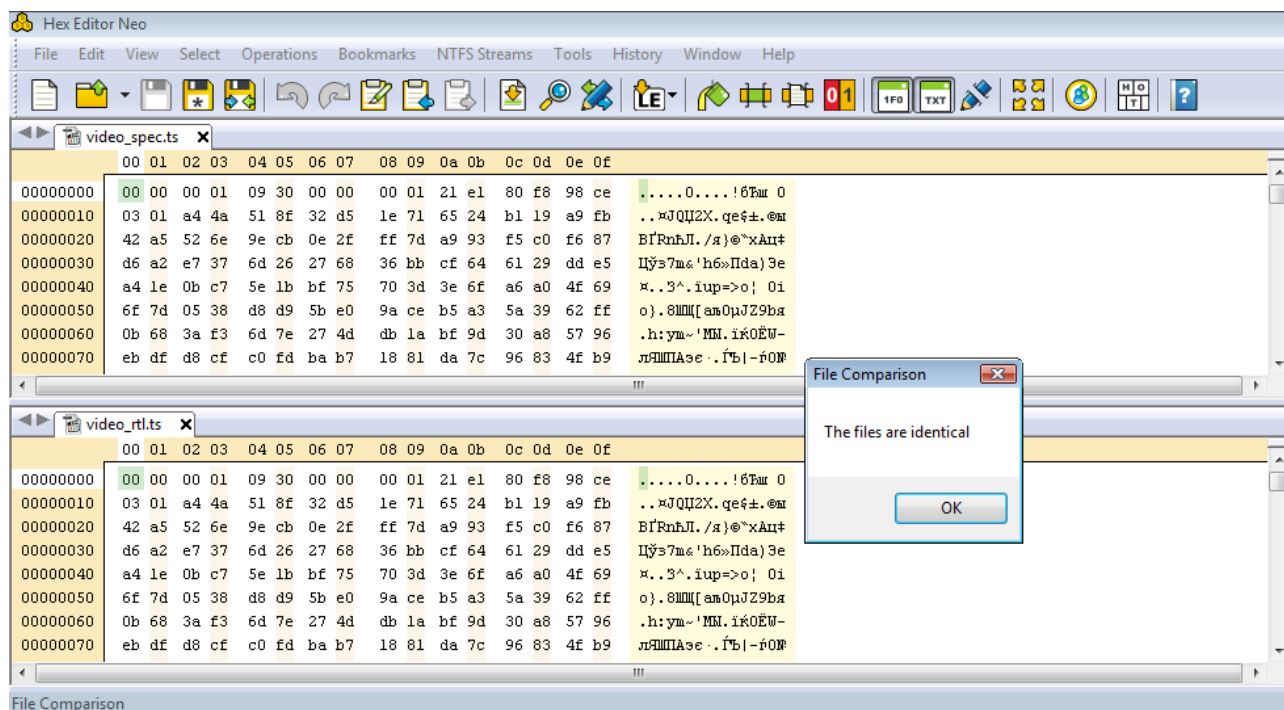


Figura 51: Comparação dos arquivos de saída de vídeo da especificação de alto nível e da implementação RTL.

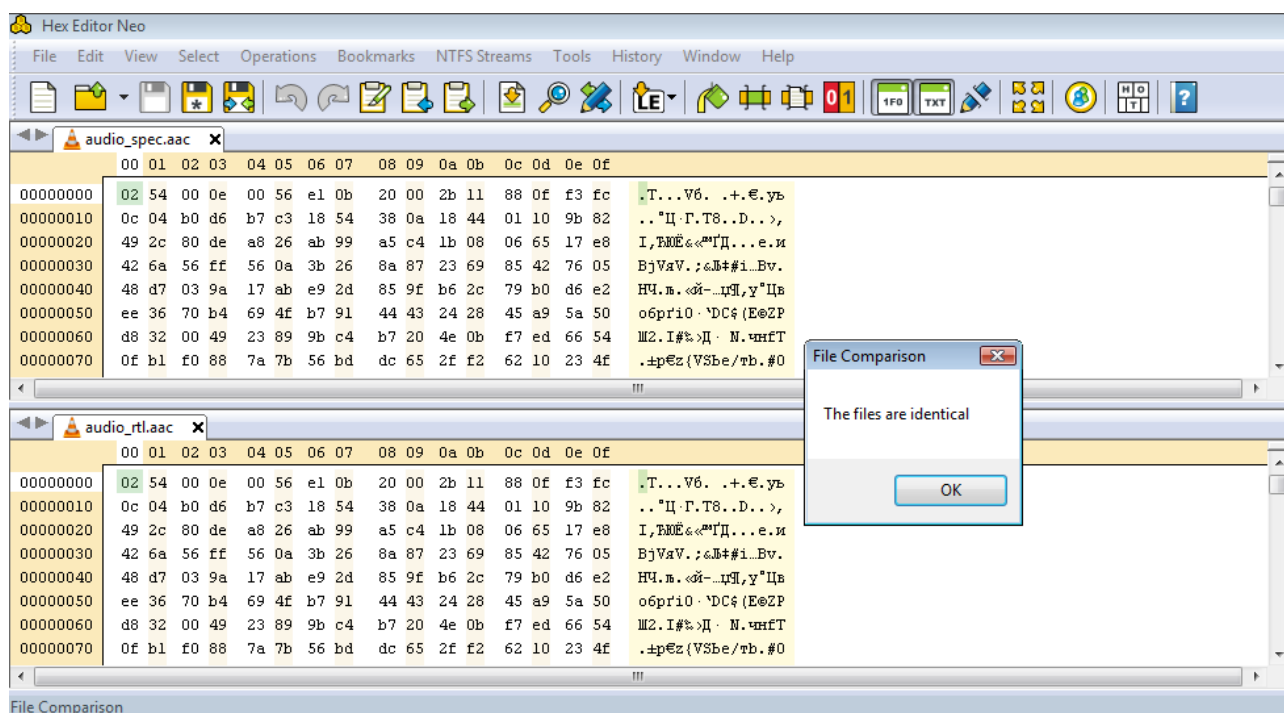


Figura 52: Comparação dos arquivos de saída de áudio da especificação de alto nível e da implementação RTL.

Finalmente os arquivos binários demultiplexados produzidos pelo modelo RTL também foram reproduzidos adequadamente pelos reprodutores de áudio e vídeo.

Somente com a validação no nível RTL, foi possível partir para a síntese lógica do demultiplexador proposto.

5.4 Síntese lógica e mapeamento tecnológico

Através de ferramentas de **síntese lógica**, foram geradas *netlists* de implementação em FPGA e em ASIC.

Para implementação FPGA, foram realizadas as seguintes tarefas:

- A produção de ***netlists* genéricas** foi realizada pela ferramenta Precision RTL Plus v2008a.39 (PRECISION RTL PLUS, 2009) para os dispositivos FPGAs Stratix – EP1S10F780C, Stratix – EP1S60F1020C, Stratix II – EP2S60F672C e Cyclone III – EP3C25F324C, da empresa Altera, disponíveis na UFPB.
- A produção de ***netlist* mapeada** foi feita pela ferramenta Quartus II 8.0 (ALTERA, 2010) apenas para o dispositivo EP2S60F672C da família Stratix II.

Para implementação ASIC, foi realizada a seguinte tarefa:

- A produção de ***netlists* mapeadas** pela ferramenta LeonardoSpectrum Level 3 v2008a.5 (LEONARDOSPECTRUM, 2009) para as tecnologias alvo ASIC disponíveis no design kit ADK 3.1 fornecido pela Mentor Graphics.

5.4.1 FPGA

O início da síntese lógica em **FPGA** do demultiplexador foi realizada pela ferramenta Precision RTL Plus para os dispositivos EP1S10F780C, EP1S60F1020C, EP2S60F672C e EP3C25F324C. A aplicação do modelo RTL na ferramenta produziu ***netlists* genéricas** (arquivos EDIF) para cada dispositivo. Neste estágio, apenas otimizações *timing-driven* RTL, como MUX e caminho de dados são feitas, parando antes

do mapeamento tecnológico.

As **informações de síntese** a seguir foram obtidos através dos relatórios da ferramenta Precision RTL Plus v2008a.39 disponível no pacote HEP da Mentor Graphics. As **medidas de qualidade** aferidas para síntese do demultiplexador proposto em FPGA estão relacionadas a **área** (instâncias) e **desempenho** (frequência máxima e *time slack*).

Tabela 21: Informações de síntese em FPGA do demultiplexador.

Dispositivo	Instâncias (<i>gate count</i>)	Freq. Max. (Mhz)	<i>Time Slack – Caminho Crítico (ns)</i>
EP1S10F780C	42702	125,93	29,06
EP1S60F1020C	42702	125,93	29,06
EP2S60F672C	42782	139,1	29,81
EP3C25F324C	42636	118,37	28,55

É possível verificar, através da tabela 21, que não existe muita diferença entre os dispositivos na questão de área e desempenho relacionado à síntese FPGA do demultiplexador proposto. A única restrição para a área de uma síntese em FPGA é a relacionada ao **fitting** no dispositivo alvo. Estas informações específicas de cada dispositivo estão detalhadas nas tabelas de resultados presentes no próximo capítulo.

A partir da *netlist* genérica gerada para o dispositivo EP2S60F672C, foi possível realizar um **mapeamento tecnológico** do demultiplexador proposto. Neste estágio, foram realizadas otimizações lógicas e o projeto foi mapeado em células descritas na biblioteca de tecnologia do dispositivo. O mapeamento tecnológico foi feito com a ferramenta Quartus II, pois é nela que está presente a biblioteca de tecnologia do dispositivo. Dessa forma, produziu-se uma **netlist mapeada** e um arquivo “.sdo” – *Standard Delay Format (SDF) Output File* – que descreve informações de temporização, usado na simulação pós-síntese do módulo especificado.

5.4.2 ASIC

A síntese lógica em **ASIC** do demultiplexador foi realizado através da ferramenta LeonardoSpectrum Level 3 v2008a.5 disponível no pacote HEP da Mentor Graphics. As

tecnologias ASIC disponíveis no design kit ADK 3.1 fornecido também pela Mentor Graphics são: TSMC 350nm, TSMC 250nm, TSMC 180nm, AMI 1.2um e AMI 500nm.

As **informações de síntese** a seguir foram obtidos através dos relatórios da ferramenta LeonardoSpectrum Level 3 v2008a.5 também disponível no pacote HEP da Mentor Graphics. A **medida de qualidade** aferida para síntese em ASIC deve ser feita tanto antes como após as fases de *placement* e *routing* e estarão relacionadas à área e aos atrasos de propagação. Devido a problemas de configuração das ferramentas e design kits provocados por compatibilidade de sistemas operacionais ainda em fase de solução no LASIC, essas últimas fases deixaram de fazer parte deste trabalho, mas estão desde já propostas como trabalho a ser realizado no LASIC tão logo os problemas sejam sanados.

Tabela 22: Informações de área para as tecnologias ASIC disponíveis no design kit ADK 3.1.

Tecnologia	Instâncias (<i>gate count</i>)
TSMC 350nm	108273
TSMC 250nm	108273
TSMC 180nm	108273
AMI 1.2um	108273
AMI 500nm	108273

Verificou-se através dos relatórios de síntese, que o *script* criado para fazer a síntese lógica através do LeonardoSpectrum, produziu o mesmo resultado de **elementos lógicos**.

Uma análise pode ser feita a respeito do número de elementos lógicos usados nos blocos do demultiplexador, a partir do relatório apresentado na figura 53. É possível calcular que o número de elementos lógicos do bloco IN_BUFFER é 27382, do bloco TRANSPORT_PACKET_ANALYSER é 21545, do bloco PES_EXTRACTOR é 24652 e do bloco PSI_SI_EXTRACTOR é 34820. Assim, é possível saber o relacionamento entre o tamanho dos blocos do demultiplexador proposto e afirmar que o maior bloco é o PSI_SI_EXTRACTOR.

Cell	Library	References	Total Area	
FALSE	PRIMITIVES	1 x 1	1 FALSE	2 2 sub_7u_7u_7u_0
OR2	PRIMITIVES	2 x 1	2 OR2	9 9 gt_8u_8u
in_buffer	work	1 x 1	1 ram_dq_da_inclock_inclock2_1504_2_4	15 15 inc_8u_8u_0
		2 2 inc_2u_2u_0		47 47 mux_4u_4u
		6 6 DFFERS		1 1 or_5u_5u
		2 2 select_4_4		3 3 sub_12u_12u_12u_0
		4 4 TRUE		3 3 select_5_5
		4 4 mux_4u_4u		11 11 select_4_4
		19 19 eq_13u_13u		5 5 eq_12u_12u
		10 10 eq_2u_2u		3 3 gt_12u_12u
		1 1 or_19u_19u		79 79 mux_8u_8u
		7 7 FALSE		3 3 inc_12u_12u_0
		4547 4547 DFFRS		1617 1617 mux_16u_16u
		10620 10620 AND2		16 16 eq_3u_3u
		36 36 INV		1 1 inc_6u_6u_0
		9073 9073 MUX		1 1 sub_5u_5u_5u_0
		3050 3050 OR2		13 13 TRUE
pes	work	1 x 3	3 ram_dq_da_inclock_inclock2_8_8_256	15 15 FALSE
		54 54 DFFERS		7781 7781 DFFRS
		6 6 TRUE		22 22 eq_8u_8u
		7 7 FALSE		17147 17147 AND2
		1588 1588 DFFRS		4531 4531 OR2
		78 78 INV		104 104 INV
		8 8 eq_8u_8u		1509 1509 select_3_3
		49 49 OR2		1803 1803 MUX
		42 42 select_3_3		32 32 eq_4u_4u
		1508 1508 mux_32u_32u		1 1 or_6u_6u
		1 1 add_8u_8u_8u_0_0	transport_packet work	1 x 2 2 and_5u_5u
		5 5 eq_2u_2u		8 8 eq_3u_3u
		19061 19061 MUX		2 2 mux_8u_8u
		3 3 gt_8u_8u		1 1 ram_dq_da_inclock_inclock2_8_8_256
		9 9 inc_8u_8u_0		18 18 DFFERS
		1 1 inc_7u_7u_0		2 2 eq_8u_8u
		1 1 eq_24u_24u		5 5 select_4_4
		1 1 gt_7u_7u		3014 3014 mux_4u_4u
		2145 2145 AND2		5 5 inc_8u_8u_0
		16 16 select_4_4		4 4 gt_8u_8u
		3 3 eq_13u_13u		8 8 TRUE
		1 1 select_5_5		8 8 FALSE
		4 4 add_6u_6u_6u_0		41 41 INV
		4 4 add_7u_7u_7u_0		4589 4589 DFFRS
		16 16 XOR2		48 48 OR2
		2 2 inc_3u_3u_0		19 19 eq_13u_13u
		2 2 inc_6u_6u_0		4798 4798 AND2
		32 32 eq_5u_5u		8964 8964 MUX
		1 1 or_8u_8u		8 8 eq_2u_2u
		1 1 or_25u_25u		1 1 or_14u_14u
psi_si	work	1 x 2	2 ram_dq_da_inclock_inclock2_8_8_256	
		36 36 DFFERS		
		8 8 eq_2u_2u		
				Number of ports : 91
				Number of nets : 3132
				Number of instances : 7
				Number of references to this view : 0

Figura 53: Relatório de área do ts_demux.

5.4.3 Validação no nível das portas (gate level)

A verificação neste nível utilizou os mesmos *testbenches* e vetores de teste do nível RTL. O TS foi aplicado à especificação de alto-nível e à implementação lógica do dispositivo Stratix II – EP2S60F672C.

Decidiu-se, a seguir, fazer uma **análise de temporização** para a Stratix II, já que, como apresentado posteriormente no capítulo 6 (resultados, etc), foi possível seguir com este dispositivo com bastante folga no quesito área.

No quesito desempenho, considera-se uma taxa de bit de entrada máxima de 120 Mbits/segundo para o TS de entrada, onde a frequência de operação para a arquitetura proposta deve ser de pelo menos 15 Mhz para atender tal taxa.

A partir das informações de desempenho apresentadas na tabela 21, foi definido a frequência de operação inicial máxima do relógio de 100 Mhz para o mapeamento tecnológico realizado pela ferramenta Quartus II. Deste modo, tem-se uma folga de aproximadamente 25% em relação às frequências máximas relatadas pela ferramenta Precision RTL Plus.

A partir da opção TimingQuest Timing Analyser, presente no Quartus II 8.0 da Altera, foram gerados relatórios para a *netlist* criada. Utilizando o modelo de *delay slow*, o resultado da frequência máxima foi de 114,34 Mhz. Para o pior caso, o *time slack* de *setup* foi de 1,254 ns e o *time slack* de *hold* foi de 0,341 ns. O detalhamento dos relatórios dos tempos de *setup*, dos tempos de *hold*, dos tempos de relógio para saída e dos tempos mínimos de relógio para saída são apresentados no anexo A.

No entanto ao realizar-se a **simulação pós-síntese** – figura 54 – utilizando o mesmo teste da validação do RTL, utilizando o arquivo “.sdo” – *Standard Delay Format (SDF) Output File* – criado pelo Quartus II, que anota o atraso dos elementos nos arquivos a serem simulados (*backannotation*), verificou-se que alguns tempos de *setup* e *hold* foram violados utilizando a frequência de 100 Mhz. Passou-se a trabalhar com a frequência diminuída de 50 Mhz (400 Mbits/segundo) na simulação pós-síntese, e não houve mais violações.

Dessa forma, pode-se afirmar que com a síntese realizada é possível operar a uma frequência onde a taxa de bit de entrada máxima de 120 Mbits/segundo para o TS é atendida, abrindo possibilidade de se trabalhar com múltiplos TS's simultaneamente.

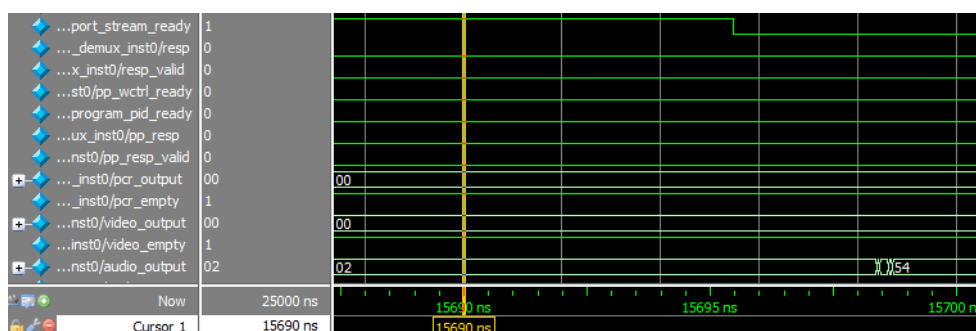


Figura 54: Simulação pós-síntese da *netlist* do TS_DEMUX para Stratix II.

Os arquivos de saída de áudio e vídeo da especificação de alto-nível e da simulação *gate-level* foram também comparados pela ferramenta HDD Hex Editor Neo

v4.93, usando a mesma opção do método de comparação de arquivos que destaca a diferença entre eles.

Assim, foi possível também atestar a igualdade de ambos arquivos, e afirmar que a implementação do demultiplexador foi validada e que alcançou níveis adequados de temporização.

6 Resultados de Mapeamento e Avaliação

Este último capítulo apresenta e discute os **resultados alcançados** com a implementação do módulo proposto. São resultados da etapa de síntese lógica em FPGA.

6.1 Resultados de Mapeamento Tecnológico em FPGA

- Stratix – EP1S10F780C

Verifica-se, na tabela 23, que não é possível realizar o *fitting* do demultiplexador proposto na Stratix EP1S10F780C. O número de células lógicas necessárias (38877) é maior que os número de células lógicas disponíveis (10570) neste dispositivo. Logo, uma possível prototipação utilizando este dispositivo não é possível.

Tabela 23: Informações de utilização do dispositivo EP1S10F780C.

Recursos	Usados	Disponíveis	Utilização
E/S	91	427	21,31%
Células Lógicas	38877	10570	367,81%
Bits de Memória	18236	920448	1,98%
Bloco DSP elem 9-bit	0	48	0,00%

- Stratix – EP1S60F1020C

Verifica-se, na tabela 24, que, para a Stratix EP1S60F1020C, já é possível realizar o *fitting* do demultiplexador proposto. O número de células lógicas necessárias (38877), neste caso, é menor que os número de células lógicas disponíveis (57120). E é possível verificar também que os outros recursos disponíveis são maiores que os necessários. O recurso de células lógicas possui uma taxa de utilização razoável, enquanto que os outros recursos são sub-utilizados, devido à baixa taxa de utilização. Uma possível prototipação utilizando este dispositivo é possível.

Tabela 24: Informações de utilização do dispositivo EP1S60F1020C.

Recursos	Usados	Disponíveis	Utilização
E/S	91	782	11,64%
Células Lógicas	38877	57120	68,06%
Bits de Memória	18236	5215104	0,35%
Bloco DSP elem 9-bit	0	144	0,00%

- Stratix II – EP2S60F672C

Na tabela 25, verifica-se que é possível realizar o *fitting* do demultiplexador proposto na Stratix II EP2S60F672C. É possível verificar que todos os recursos disponíveis são maiores que os necessários. Neste caso, como se pode observar, todos os recursos são sub-utilizados. Uma possível prototipação utilizando a Stratix II EP2S60F672C é possível.

Tabela 25: Informações de utilização do dispositivo EP2S60F672C.

Recursos	Usados	Disponíveis	Utilização
E/S	91	493	18,46%
LUT	10257	48352	21,21%
Registrador	20039	51182	39,15%
Bits de Memória	18236	2544192	0,72%
Bloco DSP elem 9-bit	0	288	0,00%

- Ciclone III – EP3C25F324C

Na tabela 26, verifica-se que é também possível realizar o *fitting* do demultiplexador proposto no Ciclone III EP3C25F324C. Verifica-se, também, que todos os recursos disponíveis são maiores que os necessários. As taxa de utilização dos recursos se mostra mais bem equilibrada que a dos outros dispositivos testados. Uma possível prototipação utilizando o Ciclone III EP3C25F324C também é possível.

Tabela 26: Informações de utilização do dispositivo EP3C25F324C.

Recursos	Usados	Disponíveis	Utilização
E/S	91	216	42,13%
LUT	19226	24624	78,08%
Registrador	20033	24624	81,36%
Bits de Memória	18236	608256	3,00%
Bloco DSP elem 9-bit	0	132	0,00%

6.2 Avaliação dos Resultados Obtidos

Como o verificado na tabela 23, não é possível mapear o projeto do demultiplexador na Stratix EP1S10F780C. Diferentemente deste primeiro, para os outros dispositivos, este mapeamento é possível. Assim, um protótipo em FPGA do demultiplexador proposto pode ser construído utilizando qualquer um dos dispositivos apresentados onde foi possível realizar o *fitting*.

Com o exposto, fica demonstrado que foi possível atingir os seguintes objetivos específicos propostos neste trabalho:

- Detalhar os requisitos estruturais e funcionais do Subistema de Fluxo de Transporte – MPEG-2 TS – em conformidade com o SBTVD.
- Desenvolver modelo em software para o Demultiplexador e um conjunto de fluxos TS capaz de validar sua conformidade com a especificação do SBTVD .
- Implementar e validar prova de conceito de um IP de demultiplexador MPEG-2 TS no nível RTL.
- Prototipagem com mapeamento tecnológico em FPGA.

A prototipagem com mapeamento tecnológico em ASIC, conforme explanado acima, foi parcial, no entanto ficou principalmente demonstrada a robustez da metodologia utilizada e que a conclusão das etapas que restam até a produção de uma descrição completa até o nível do silício é uma questão importante, mas que não compromete a relevância do trabalho realizado.

Considerações Finais

Este trabalho tratou o desenvolvimento e validação de uma proposta arquitetural para o Subsistema de Fluxo de Transporte MPEG-2 TS – demultiplexador de fluxo de transporte – do SBTVD, desde sua especificação e modelagem alto-nível, até sua implementação no nível de mapeamento tecnológico.

Um IP de demultiplexador de transporte MPEG-2 TS, compatível com o padrão SBTVD, foi mapeado em FPGA e (parcialmente) em ASIC a partir de uma descrição Verilog HDL (aproximadamente 7800 linhas de código) que foi primeiramente modelado funcionalmente na linguagem C/C++ (aproximadamente 4200 linhas de código).

Atualmente, simples chips SoCs dedicado à recepção de TV digital integram mais de 30 milhões de transistores e 3 ou mais processadores. O IP aqui apresentado, com seus 108 mil elementos lógicos, o que representa menos de 1% do número total de componentes se integrado em tais SOC's, pode desempenhar um papel importante na busca de núcleos de processadores mais simples para embarcar em sistemas de recepção, notadamente nos sistemas móveis, na medida em que ficam liberados de fornecer suporte de software para as tarefas como a recuperação de relógio e filtragem de tabelas de seção, responsáveis por considerável carga de computação na recepção de televisão digital.

A partir do alto nível até o nível RTL e de pós-síntese com *backannotation*, foram validados com sucesso a demultiplexação de um TS de referência contendo 13080x188 bytes, o que corresponde a um fluxo multimídia de 50 segundos.

Os avanços e impactos do presente trabalho, muito além do circuito desenvolvido, podem ser enfocados sob o prisma sistêmico do Programa Nacional de Microeletrônica e do Sistema Brasileiro de TV Digital, uma vez que proporcionou um aprofundamento dos conhecimentos relativos às estruturas de multiplexagem/de-multiplexagem dos fluxos de sinais, dados, informações, aplicações que são tratados no padrão SBTVD.

A partir de uma aplicação estratégica para o Sistema Brasileiro de TV Digital, este trabalho de mestrado proporcionou capacitação para o estado da arte no fluxo de projeto de um circuito integrado complexo.

Ao conseguir contemplar no plano de trabalho principal a inserção de um período

de treinamento na fase 1 do projeto CI-Brasil, foi possível ao mestrando introduzir-se e aprofundar-se no uso de ferramentas de auxílio ao projeto de CIs que compõem ambientes industriais.

No ambiente da UFPB, este trabalho foi conduzido ao mesmo tempo que seis projetistas ainda em curso de sua graduação desenvolviam um *chip* mais simples dentro do projeto *Brazil-IP*. Procedimentos metodológicos adotados no LASIC, somados aos do *Brazil-IP* (*IP-Process*) contribuíram na consolidação da proposta metodológica aqui exposta, proposta essa que, juntamente com o circuito apresentado servirão de alicerce para as futuras gerações de desenvolvedores e pesquisadores que serão formados no LASIC/LASID e seus parceiros.

Como continuação direta deste trabalho, para trabalhos futuros, pode-se citar a realização das etapas de síntese, posicionamento e roteamento do demultiplexador, até a elaboração de um leiaute físico.

Adicionalmente, é também de grande interesse a criação de um ambiente de gestão de projetos que envolva o funcionamento do ambiente CAD, da metodologia de projeto do *chip*.

Sugere-se ainda para o *middleware* Ginga, que vem sendo desenvolvido na UFPB integralmente como uma camada de software, a busca de blocos passíveis de serem implementados em hardware.

Referências Bibliográficas

ABNT NBR 15602-3, Norma Brasileira. **Televisão digital terrestre – Codificação de vídeo, áudio e multiplexação – Parte 3: Sistemas de multiplexação de sinais.** 2007.

ABNT NBR 15603-1, Norma Brasileira. **Televisão digital terrestre – Multiplexação e serviços de informação (SI) – Parte 1: SI do sistema de radiodifusão.** 2007.

ABNT NBR 15603-2, Norma Brasileira. **Televisão digital terrestre – Multiplexação e serviços de informação (SI) – Parte 2: Estrutura de dados e definições da informação básica de SI.** 2007.

ABNT NBR 15603-3, Norma Brasileira. **Televisão digital terrestre – Multiplexação e serviços de informação (SI) – Parte 3: Sintaxes e definições de informação estendida do SI.** 2007.

ABNT NBR 15604, Norma Brasileira. **Televisão digital terrestre – Receptores.** 2007.

ALTERA. **Altera.** Disponível em: <<http://www.altera.com>>. Acesso em: fev. 2010.

ARIB STD-B10, ARIB Standard. **Service information for digital broadcasting system.** 2008.

ARM IHI 0022B. **AMBA AXI Protocol.** 2004.

BERGERON, Janick. **Writing Testbenches using SystemVerilog.** Editora Springer. 2006.

BRIDGEWATER, Kevin E. DEISS, Michael S. **Grand Alliance Transport Stream Demultiplexer Design & Implementation**. Proceedings of International Conference on Consumer Eletronics, Junho, 1995.

DATATYPES. **Algorithmic C Data Types**. Disponível em: <http://www.mentor.com/products/esl/high_level_synthesis/ac_datatypes>. Acesso em: mai. 2008.

DE MICHELI, Giovanni. **Synthesis and optimization of digital circuits**. Editora McGraw-Hill. 1994.

DUTTA, S. JENSEN, R. RIECKMANN, A. **Viper: A Multiprocessor SoC for Advanced Set-Top Box and Digital TV Systems**. IEEE Design & Test of Computers, Vol. 18, No. 5, p. 21-31, Setembro-Outubro, 2001.

ETSI EN 300 468, European Standard. **Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB systems**. 2003

GAJSKI, D. D. **IP-Based Design Methodology**. Annual ACM IEEE Design Automation Conference, p.43, Junho, 1999.

GAJSKI, D. D. WU, A. C. -H. CHAIYAKUL, V. et al. **Essential issues for IP reuse**. Design Automation Conference, Proceedings of the Asia and South Pacific, p. 37-42, Janeiro, 2000.

GCC. **GCC, the GNU Compiler Collection**. Disponível em: <<http://gcc.gnu.org/>>. Acesso em: mai. 2008.

HANINTAK. **IPTV STB – HW**. Disponível em: <http://onyx.yonsei.ac.kr/~tta2007/files/07_11_08_02.pdf>. Acesso em: nov. 2008.

HANNA, C. GILLIES, D. COCHON, et al. **Demultiplexer IC for MPEG2 Transport Streams**. IEEE Transactions on Consumer Electronics, Vol. 41, No. 3, Agosto, 1995.

HARRIS, David Money. HARRIS, Sarah L. **Digital Design and Computer Architecture**. Editora Morgan Kaufmann. 2007.

IBM39MPEGCS24PFA16C IBM39MPEGCS24PFDA16C, Datasheet. **MPEG CS24 High Performance Audio/Video Decoder**. 1999.

INAMORI, M. NAGANUMA, J. WAKABAYASHI, H. ENDO, M. **A Memory-based Architecture for MPEG2 System Protocol LSIs**. Proceedings of European Design and Test Conference, Março, 1996.

ISO/IEC 13818-1, **International Standard. Information technology – Generic of coding of moving pictures and associated audio information – Part 1: Systems**. 2000.

KEATING, M. BRICAUD, P. **Reuse Methodology Manual for System-on-a-Chip Designs**. New York: Kluwer Academic Publishers, 2002.

LEONARDOSPECTRUM. **LeonardoSpectrum**. Disponível em: <http://www.mentor.com/products/fpga/synthesis/leonardo_spectrum/>. Acesso em: out. 2009.

MADISETTI, V. K. SHEN L. **Interface Design for Core-Based Systems**. IEEE Design &

Test of Computers, p. 42-51, Outubro-Dezembro, 1997.

OLIVEIRA, M. F. S. BRISOLARA, L. B. WAGNER F. R. CARRO, L. **Embedded SW Design Exploration Using UML-based Estimation Tools**. UML FOR SoC DESIGN (UML-SOC'05), 2005, Anaheim. 2005.

PALMA, J. C. S. MORAES, F. G. CALAZANS, N. L. V. **Métodos para Desenvolvimento e Distribuição de IP Cores**. Seminário de Computação Reconfigurável, Belo Horizonte, 2001.

PRECISION RTL PLUS. **Precision RTL Plus**. Disponível em : <http://www.mentor.com/products/fpga/synthesis/precision_rtl_plus/>. Acesso em: out. 2009.

QUEIROZ, D. C. **Implementação do barramento on-chip AMBA baseada em computação reconfigurável**. 2005. Dissertação (Mestrado em Ciências da Computação e Matemática Computacional) – Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação, USP, São Carlos.

QUESTA. **Questa Advanced Verification and Debug Technologies**. Disponível em: <<http://www.mentor.com/products/fv/questa/>>. Acesso em: out. 2009.

S5H2010 S3C2800, Samsung. **Chipset Solution for High-Definition TVs and Sep-Top Boxes**. 2003.

SAA7205H, Data sheet. **MPEG-2 systems demultiplexer**. 1997.

SAA7214, Data Sheet. **Transport MPEG2 source decoder**. 2001.

SAA7219, Data Sheet. **MPEG2 Transport RISC processor**. 2001.

SAA7240, Data Sheet. **MPEG-2 Transport RISC processor**. 2001.

SCHAUMONT, P. CMAR, R. VERNALDE, S. et al. **Hardware Reuse at the Behavioral Level**. Annual ACM IEEE Design Automation Conference, Proceedings of the 36th ACM/IEEE conference on Design automation, p. 784-789, 1999.

STB7100. **Low-cost HDTV set-top box decoder for H264/AVC and MPEG-2**. 2005.

STI7101. **Low-cost HDTV set-top box decoder for H.264**. 2008.

THOMAS, T. **Technology for IP Reuse and Portability**. IEEE Design & Test of Computers, Vol. 16, No. 4, p. 7-13, Outubro-Dezembro, 1999.

TSREADER. **TSReader**. Disponível em: <<http://www.tsreader.com/tsreader/index.html>>. Acesso em: mai. 2009.

VIDEOLAN. **VLC Media Player**. Disponível em: <<http://www.videolan.org/vlc/>>. Acesso em: out. 2008.

WAGNER, F. R. CESÁRIO, W. JERRAYA, A. A. **Hardware/Software IP Integration Using the ROSES Design Environment**. ACM Transactions on Embedded Computing Systems, Vol. 6, No. 3, Julho, 2007.

WAGNER, F. R. CESARIO, W. O. CARRO, L. JERRAYA, A. A. **Strategies for the Integration of Hardware and Software IP Components in Embedded Systems-on-**

Chip. Integration, The VLSI Journal. Vol. 37, No. 4, p. 223-252, Setembro, 2004.

WEST, N.H.E. HARRIS, D. **CMOS VLSI Design : A circuit and system perspective.** 3ª Edição. Editora Pearson. 2005.

ZHANG, Chunrong. ZHENG, Shibao. WANG, Feng. YUAN, Chi. **Design and Implementation of Transport Stream Demultiplexer in HDTV Decoder SoC.** IEEE Transactions on Consumer Electronics, Vol. 51, No. 2, Maio, 2005.

Anexo A

Tabela 27: Relatório de tempos de setup da netlist do ts_demux para Stratix II.

Data port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
audio_rd_ena	clock	3574	3574	Rise	clock
data_rd_ena	clock	4042	4042	Rise	clock
mpeg_transport_stream[*]	clock	2961	2961	Rise	clock
mpeg_transport_stream[0]	clock	2716	2716	Rise	clock
mpeg_transport_stream[1]	clock	2558	2558	Rise	clock
mpeg_transport_stream[2]	clock	2812	2812	Rise	clock
mpeg_transport_stream[3]	clock	2581	2581	Rise	clock
mpeg_transport_stream[4]	clock	2961	2961	Rise	clock
mpeg_transport_stream[5]	clock	2775	2775	Rise	clock
mpeg_transport_stream[6]	clock	2613	2613	Rise	clock
mpeg_transport_stream[7]	clock	2773	2773	Rise	clock
mpeg_transport_stream_last	clock	3590	3590	Rise	clock
mpeg_transport_stream_valid	clock	7585	7585	Rise	clock
pcr_rd_ena	clock	3568	3568	Rise	clock
pp_wctrl	clock	2568	2568	Rise	clock
pp_wctrl_valid	clock	2873	2873	Rise	clock
program_pid[*]	clock	2590	2590	Rise	clock
program_pid[0]	clock	2433	2433	Rise	clock

Data port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
program_pid[1]	clock	2243	2243	Rise	clock
program_pid[2]	clock	2241	2241	Rise	clock
program_pid[3]	clock	2320	2320	Rise	clock
program_pid[4]	clock	2590	2590	Rise	clock
program_pid[5]	clock	2387	2387	Rise	clock
program_pid[6]	clock	2226	2226	Rise	clock
program_pid[7]	clock	2291	2291	Rise	clock
program_pid[8]	clock	2225	2225	Rise	clock
program_pid[9]	clock	2358	2358	Rise	clock
program_pid[10]	clock	2495	2495	Rise	clock
program_pid[11]	clock	2333	2333	Rise	clock
program_pid[12]	clock	2267	2267	Rise	clock
program_pid_last	clock	2696	2696	Rise	clock
program_pid_valid	clock	3337	3337	Rise	clock
psi_si_rd_ena	clock	3335	3335	Rise	clock
rst_n	clock	9579	9579	Rise	clock
video_rd_ena	clock	2839	2839	Rise	clock
wctrl	clock	2155	2155	Rise	clock
wctrl_valid	clock	2115	2115	Rise	clock

Tabela 28: Relatório de tempos de hold da netlist do ts_demux para Stratix II.

Data port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
audio_rd_ena	clock	-2227	-2227	Rise	clock
data_rd_ena	clock	-2701	-2701	Rise	clock
mpeg_transport_stream[*]	clock	-2319	-2319	Rise	clock
mpeg_transport_stream[0]	clock	-2477	-2477	Rise	clock
mpeg_transport_stream[1]	clock	-2319	-2319	Rise	clock
mpeg_transport_stream[2]	clock	-2573	-2573	Rise	clock
mpeg_transport_stream[3]	clock	-2342	-2342	Rise	clock
mpeg_transport_stream[4]	clock	-2722	-2722	Rise	clock
mpeg_transport_stream[5]	clock	-2536	-2536	Rise	clock
mpeg_transport_stream[6]	clock	-2374	-2374	Rise	clock
mpeg_transport_stream[7]	clock	-2534	-2534	Rise	clock
mpeg_transport_stream_last	clock	-2154	-2154	Rise	clock
mpeg_transport_stream_valid	clock	-2124	-2124	Rise	clock
pcr_rd_ena	clock	-2228	-2228	Rise	clock
pp_wctrl	clock	-2329	-2329	Rise	clock
pp_wctrl_valid	clock	-2469	-2469	Rise	clock
program_pid[*]	clock	-1986	-1986	Rise	clock
program_pid[0]	clock	-2194	-2194	Rise	clock

Data port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
program_pid[1]	clock	-2004	-2004	Rise	clock
program_pid[2]	clock	-2002	-2002	Rise	clock
program_pid[3]	clock	-2081	-2081	Rise	clock
program_pid[4]	clock	-2351	-2351	Rise	clock
program_pid[5]	clock	-2148	-2148	Rise	clock
program_pid[6]	clock	-1987	-1987	Rise	clock
program_pid[7]	clock	-2052	-2052	Rise	clock
program_pid[8]	clock	-1986	-1986	Rise	clock
program_pid[9]	clock	-2119	-2119	Rise	clock
program_pid[10]	clock	-2256	-2256	Rise	clock
program_pid[11]	clock	-2094	-2094	Rise	clock
program_pid[12]	clock	-2028	-2028	Rise	clock
program_pid_last	clock	-2457	-2457	Rise	clock
program_pid_valid	clock	-1878	-1878	Rise	clock
psi_si_rd_ena	clock	-2029	-2029	Rise	clock
rst_n	clock	-1798	-1798	Rise	clock
video_rd_ena	clock	-1948	-1948	Rise	clock
wctrl	clock	-1916	-1916	Rise	clock
wctrl_valid	clock	-1690	-1690	Rise	clock

Tabela 29: Relatório de tempos de relógio para saída da netlist do ts_demux para Stratix

II.

Data port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
audio_empty	clock	6153	6153	Rise	clock
audio_output[*]	clock	8716	8716	Rise	clock
audio_output[0]	clock	8550	8550	Rise	clock
audio_output[1]	clock	8637	8637	Rise	clock
audio_output[2]	clock	8447	8447	Rise	clock
audio_output[3]	clock	8426	8426	Rise	clock
audio_output[4]	clock	8716	8716	Rise	clock
audio_output[5]	clock	8498	8498	Rise	clock
audio_output[6]	clock	8399	8399	Rise	clock
audio_output[7]	clock	8431	8431	Rise	clock
data_empty	clock	7372	7372	Rise	clock
data_output[*]	clock	10297	10297	Rise	clock
data_output[0]	clock	9148	9148	Rise	clock
data_output[1]	clock	9268	9268	Rise	clock
data_output[2]	clock	10195	10195	Rise	clock
data_output[3]	clock	9030	9030	Rise	clock
data_output[4]	clock	8996	8996	Rise	clock
data_output[5]	clock	10100	10100	Rise	clock
data_output[6]	clock	10297	10297	Rise	clock
data_output[7]	clock	9248	9248	Rise	clock
mpeg_transport_stream_ready	clock	5474	5474	Rise	clock
pcr_empty	clock	6038	6038	Rise	clock
pcr_output[*]	clock	9811	9811	Rise	clock
pcr_output[0]	clock	9811	9811	Rise	clock
pcr_output[1]	clock	9722	9722	Rise	clock
pcr_output[2]	clock	9663	9663	Rise	clock
pcr_output[3]	clock	9259	9259	Rise	clock
pcr_output[4]	clock	8810	8810	Rise	clock

Data port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
pcr_output[5]	clock	9589	9589	Rise	clock
pcr_output[6]	clock	9428	9428	Rise	clock
pcr_output[7]	clock	9555	9555	Rise	clock
pp_resp	clock	5881	5881	Rise	clock
pp_resp_valid	clock	5891	5891	Rise	clock
pp_wctrl_ready	clock	7672	7672	Rise	clock
program_pid_ready	clock	5915	5915	Rise	clock
psi_si_empty	clock	5634	5634	Rise	clock
psi_si_output[*]	clock	8600	8600	Rise	clock
psi_si_output[0]	clock	8600	8600	Rise	clock
psi_si_output[1]	clock	8344	8344	Rise	clock
psi_si_output[2]	clock	8415	8415	Rise	clock
psi_si_output[3]	clock	8068	8068	Rise	clock
psi_si_output[4]	clock	8469	8469	Rise	clock
psi_si_output[5]	clock	8349	8349	Rise	clock
psi_si_output[6]	clock	8427	8427	Rise	clock
psi_si_output[7]	clock	8035	8035	Rise	clock
resp	clock	5521	5521	Rise	clock
resp_valid	clock	5521	5521	Rise	clock
video_empty	clock	5484	5484	Rise	clock
video_output[*]	clock	8898	8898	Rise	clock
video_output[0]	clock	7964	7964	Rise	clock
video_output[1]	clock	8728	8728	Rise	clock
video_output[2]	clock	8215	8215	Rise	clock
video_output[3]	clock	8898	8898	Rise	clock
video_output[4]	clock	8311	8311	Rise	clock
video_output[5]	clock	8746	8746	Rise	clock
video_output[6]	clock	8441	8441	Rise	clock
video_output[7]	clock	8596	8596	Rise	clock
wctrl_ready	clock	5473	5473	Rise	clock

Tabela 30: Relatório de tempos mínimos de relógio para saída da netlist do ts_demux para Stratix II.

Data port	Clock Port	Rise	Fall	Clock Edge	Clock Reference	Data port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
audio_empty	clock	6153	6153	Rise	clock	pcr_output[6]	clock	6199	6199	Rise	clock
audio_output[*]	clock	7106	7106	Rise	clock	pcr_output[7]	clock	6458	6458	Rise	clock
audio_output[0]	clock	7106	7106	Rise	clock	pp_resp	clock	5881	5881	Rise	clock
audio_output[1]	clock	7270	7270	Rise	clock	pp_resp_valid	clock	5891	5891	Rise	clock
audio_output[2]	clock	7289	7289	Rise	clock	pp_wctrl_ready	clock	7672	7672	Rise	clock
audio_output[3]	clock	7528	7528	Rise	clock	program_pid_ready	clock	5915	5915	Rise	clock
audio_output[4]	clock	7805	7805	Rise	clock	psi_si_empty	clock	5634	5634	Rise	clock
audio_output[5]	clock	7566	7566	Rise	clock	psi_si_output[*]	clock	6521	6521	Rise	clock
audio_output[6]	clock	7515	7515	Rise	clock	psi_si_output[0]	clock	7157	7157	Rise	clock
audio_output[7]	clock	7111	7111	Rise	clock	psi_si_output[1]	clock	6786	6786	Rise	clock
data_empty	clock	7372	7372	Rise	clock	psi_si_output[2]	clock	6781	6781	Rise	clock
data_output[*]	clock	7076	7076	Rise	clock	psi_si_output[3]	clock	6862	6862	Rise	clock
data_output[0]	clock	7531	7531	Rise	clock	psi_si_output[4]	clock	6731	6731	Rise	clock
data_output[1]	clock	7515	7515	Rise	clock	psi_si_output[5]	clock	6799	6799	Rise	clock
data_output[2]	clock	7714	7714	Rise	clock	psi_si_output[6]	clock	6742	6742	Rise	clock
data_output[3]	clock	7076	7076	Rise	clock	psi_si_output[7]	clock	6521	6521	Rise	clock
data_output[4]	clock	7237	7237	Rise	clock	resp	clock	5521	5521	Rise	clock
data_output[5]	clock	7800	7800	Rise	clock	resp_valid	clock	5521	5521	Rise	clock
data_output[6]	clock	7849	7849	Rise	clock	video_empty	clock	5484	5484	Rise	clock
data_output[7]	clock	7117	7117	Rise	clock	video_output[*]	clock	5840	5840	Rise	clock
mpeg_transport_stream_ready	clock	5474	5474	Rise	clock	video_output[0]	clock	6138	6138	Rise	clock
pcr_empty	clock	6038	6038	Rise	clock	video_output[1]	clock	6508	6508	Rise	clock
pcr_output[*]	clock	6199	6199	Rise	clock	video_output[2]	clock	6503	6503	Rise	clock
pcr_output[0]	clock	6529	6529	Rise	clock	video_output[3]	clock	6483	6483	Rise	clock
pcr_output[1]	clock	6847	6847	Rise	clock	video_output[4]	clock	5840	5840	Rise	clock
pcr_output[2]	clock	6463	6463	Rise	clock	video_output[5]	clock	6446	6446	Rise	clock
pcr_output[3]	clock	6399	6399	Rise	clock	video_output[6]	clock	6354	6354	Rise	clock
pcr_output[4]	clock	6543	6543	Rise	clock	video_output[7]	clock	6403	6403	Rise	clock
pcr_output[5]	clock	6544	6544	Rise	clock	wctrl_ready	clock	5473	5473	Rise	clock